

**VILNIAUS UNIVERSITETAS**  
**MATEMATIKOS IR INFORMATIKOS FAKULTETAS**  
**DUOMENŲ MOKSLO STUDIJŲ PROGRAMA**

**Duomenų mokslo projektas - kursinis darbas**

**Socialinių tinklų įtaka kriptovalutų  
kainoms**

**Social media impact on cryptocurrencies  
prices**

Darbo atliko: Matas Gaulia,  
Jekaterina Sergejeva  
Darbo vadovas: Linas Petkevičius

**VILNIUS 2022**

# Turinys

<b>1</b>	<b>Įvadas</b>	<b>6</b>
<b>2</b>	<b>Temos aprašymas</b>	<b>7</b>
<b>3</b>	<b>Pirminė duomenų analizė</b>	<b>9</b>
<b>4</b>	<b>Eksperimentai</b>	<b>13</b>
4.1	Dieniniai grafikai . . . . .	13
4.2	Valandiniai grafikai . . . . .	14
4.3	Minutiniai grafikai . . . . .	16
4.4	Išvados apie ARIMA modelį . . . . .	17
<b>5</b>	<b>Duomenų paruošimas</b>	<b>18</b>
<b>6</b>	<b>Tiesinė regresija</b>	<b>19</b>
6.1	Dieniniai grafikai . . . . .	19
6.2	Valandiniai grafikai . . . . .	19
6.3	Minutiniai grafikai . . . . .	19
<b>7</b>	<b>Pycaret biblioteka</b>	<b>21</b>
7.1	Dieniniai grafikai . . . . .	21
7.2	Valandiniai grafikai . . . . .	23
7.3	Minutiniai grafikai . . . . .	25
<b>8</b>	<b>LSTM modelio sudarymas ir tyrimas</b>	<b>27</b>
8.1	Dieniniai duomenys . . . . .	27
8.2	Valandiniai duomenys . . . . .	28
8.3	Minutiniai duomenys . . . . .	30
8.4	Išvados apie LSTM modelį . . . . .	31
<b>9</b>	<b>Išvados bei rekomendacijos</b>	<b>32</b>
<b>10</b>	<b>Priedai</b>	<b>34</b>
10.1	Duomenų valymas . . . . .	34
10.2	ARIMA modelis . . . . .	37
10.3	Sentimento modelis . . . . .	41
10.4	Duomenų paruošimas . . . . .	47
10.5	Tiesinė regresija . . . . .	49
10.6	Pycaret paketas . . . . .	52
10.7	LSTM modelis . . . . .	56

## Lentelių sąrašas

1	LSTM modelio parametrai . . . . .	27
2	Dieninių duomenų modelių palyginimas . . . . .	32
3	Valandinių duomenų modelių palyginimas . . . . .	32
4	Minutinių duomenų modelių palyginimas . . . . .	32

## Iliustracijų sąrašas

1	Dieniniai kainų duomenys . . . . .	9
2	Valandiniai kainų duomenys . . . . .	10
3	Minutiniai kainų duomenys . . . . .	10
4	Įrašų sentimento histograma . . . . .	11
5	Dieninis irašų skaičius pagal sentimento dominavimą . . . . .	12
6	Valandinis irašų skaičius pagal sentimento dominavimą . . . . .	12
7	Minutinis irašų skaičius pagal sentimento dominavimą . . . . .	12
8	Dieninių duomenų dekompozicija . . . . .	13
9	Dieninių duomenų liekanų pasiskirstymas . . . . .	14
10	Dieninių duomenų ARIMA rezultatai . . . . .	14
11	Valandinių duomenų dekompozicija . . . . .	14
12	Valandinių duomenų liekanų pasiskirstymas . . . . .	15
13	Valandinių duomenų ARIMA rezultatai . . . . .	15
14	Minutinių duomenų dekompozicija . . . . .	16
15	Minutinių duomenų liekanų pasiskirstymas . . . . .	16
16	Minutinių duomenų ARIMA rezultatai . . . . .	17
17	Tiesinė regresija, dieniniai duomenys . . . . .	20
18	Tiesinė regresija, valandiniai duomenys . . . . .	20
19	Tiesinė regresija, minutiniai duomenys . . . . .	20
20	Modelių palyginimas, dieniniai duomenys . . . . .	21
21	Dieninių XRP kainų pokyčių prognozavimas . . . . .	22
22	Dieniniai duomenys, kovariančių reikšmingumas . . . . .	22
23	Modelių palyginimas, valandiniai duomenys . . . . .	23
24	Valandinių XRP kainų pokyčių prognozavimas . . . . .	23
25	Valandinių XRP kainų pokyčių prognozių paklaidos . . . . .	24
26	Valandiniai duomenys, kovariančių reikšmingumas . . . . .	24
27	Modelių palyginimas, minutiniai duomenys . . . . .	25
28	Minutinių XRP kainų pokyčių prognozavimas . . . . .	25
29	Minutinių XRP kainų pokyčių prognozių paklaidos . . . . .	26
30	Minutiniai duomenys, kovariančių reikšmingumas . . . . .	26
31	LSTM liekanos, dieniniai duomenys . . . . .	27
32	LSTM prognozė, dieniniai duomenys . . . . .	28
33	LSTM liekanos, valandiniai duomenys . . . . .	28
34	LSTM prognozė, valandiniai duomenys . . . . .	29
35	LSTM liekanos, minutiniai duomenys . . . . .	30
36	LSTM prognozė, minutiniai duomenys . . . . .	30

# **Socialinių tinklų įtaka kriptovaliutų kainoms**

## **Santrauka**

Kriptovaliutos daugeliu aspektų skiriasi nuo įprastų finansinių rinkų, jų kainos priklauso nuo visiškai skirtinį dalykų negu įmonių akcijų kainos. Kriptovaliutos yra decentralizuotos, nevaldomos jokių finansinių institucijų, o jų panaudojimas ribotas. Vienas iš pagrindinių dalykų, darančių įtaką kriptovaliutų kainoms yra jų populiarumas ir žmonių nuomonė apie ją, išreišksta socialiniuose tinkluose. Remiantis šia teorija bus bandoma ištirti ar socialinių tinklų įrašai turi įtakos kriptovaliutos XRP kainai.

**Raktiniai žodžiai :** Kriptovaliutos; Laiko eilutės; Natūralios kalbos apdorojimas

# **Social media influence on cryptocurrency prices**

## **Abstract**

Cryptocurrencies differ from regular financial markets in many aspects, their prices depend on totally different things than companies' stock prices. Cryptocurrencies are decentralized, not controlled by any financial institutions, their use is very limited. One of the main things that affects cryptocurrency's price is its popularity and people's opinion about it shared on social media. Based on this theory we are going to explore whether social media posts about XRP cryptocurrency impact its price.

**Key words :** Cryptocurrencies; Time series; Natural language processing

## Acronyms

**ARIMA** Autoregressive Integrated Moving Average. 1

**BTC** Bitcoin kriptovaliuta. 1

**ETH** Ethereum kriptovaliuta. 1

**GARCH** Generalized Auto Regressive Conditional Heteroskedasticity. 1

**GRU** Gated Recurrent Unit. 1

**LSTM** Long Short Term Memory. 1

**MAE** Mean Absolute Error. 1

**MAPE** Mean Absolute Percentage Error. 1

**MSLE** Mean Squared Logarithmic Error. 1

**RMSE** Root Mean Squared Error. 1

**SPY** S&P500 investicinės fondas sekantis didžiausias 500 Jungtinių Amerikos Valstijų kompanijas. 1

**SVI** Search Volume Index. 1

**TCN** Temporal Convolutional Networks. 1

**VXX** Investicinės fondas sekantis kokio stiprio kainų judėjimo tikisi žmonės pamatyti S&P500 per ateinančius 12 mėnesių. 1

**XAU** Investicinės fondas sekantis aukso kainą. 1

**XRP** Ripple kriptovaliuta. 1

# 1 Įvadas

Kriptovaliutos – tai skaitmeniniai, tik virtualioje erdvėje egzistuojantys valiutos tipai, paremti blokų grandinės (angl. *blockchain*) technologija ir leidžiantys anonimiškai atlikti įvairius internetinius mokėjimus be trečiųjų šalių tarpininkavimo. Kriptovaliutos yra decentralizuotos ir neprižiūrimos jokių finansinių institucijų. Kiekvienais metais jų atsiranda vis daugiau, kai kurios greitai išpopuliarėja ir jų kainos sparčiai auga. [AEK<sup>+</sup>21] straipsnyje rašoma, jog tyrimas, atlirkas 2017 metais, parodė, kad tuo metu apie 6 milijonus žmonių visame pasaulyje turėjo kriptovaliutą. Praėjus vos 4 metams, šis skaičius padidėjo iki 300 milijonų. Tačiau nėra aišku kas gali lemti tam tikros kriptovaliutos populiarumą ir atvirkščiai. Stebint akcijų rinką, galima nesunkiai pasakyti kurie faktoriai daro įtaką vienos ar kitos akcijų rūšies kainų kitemams. Mes manome, jog kriptovaliutų populiarumą ir jų kainų augimą gali lemti socialinių tinklų (pvz. *Twitter*) įrašai apie ją, t.y. kuo didesnis žmonių, keliančių įrašus apie tam tikros kriptovaliutos įsigijimą ir goriančių ją, tuo daugiau atsiranda naujų pirkėjų, kriptovaliuta populiarėja ir jos kaina auga. Tuo tarpu, kai žmonės rašo daug negatyvių komentarų apie kriptovaliutą, tai gali sumažinti jos kainą.

Tyrimui mes pasirinkome 1 kriptovaliutą – XRP bei įrašus apie ją iš socialinio tinklo *Twitter*.

Darbe bus panaudojamos dvi strategijos *XRP* kainų prognozavimui. Pirmiausia bus pritaikytas ARIMA modelis laiko eilutėms ir bus bandoma nuspėti *XRP* kainų pokyčius ateityje. Antras būdas, kuris atrodo veiksmingesnis, yra *XRP* kainų prognozavimas remiantis įrašais iš socialinio tinklo Twitter apie šią kriptovaliutą. Pirmiausia kiekvienas duomenų rinkinys atskirai turi būti išanalizuotas. Iš įrašų bus gautas sentimentas (teigiamas, neigiamas), šiam tikslui pasiekti bus naudojamas paruoštas Roberta modelis. Toliau bus tikrinama hipotezė, kad socialinių tinklų įrašai apie kriptovaliutą daro įtaką jos kainai.

## Tikslias

- Patikrinti ar *Twitter* socialinio tinklo įrašai apie XRP kriptovaliutą daro įtaką jos kainų pokyčiams.

## Uždaviniai

- Išrinkti geriausią laiko eilučių modelį kriptovaliutų kainoms
- Atrasti reikšmingus požymius kriptovaliutos kainų prognozavimui
- Prognozuoti kriptovaliutos kainas pasitelkiant sentimento informaciją

## 2 Temos aprašymas

Šiomis dienomis kriptovaliutų kainų prognozavimas ir analizavimas yra gan populiarai tema moksliuose darbuose, juk pavykus sėkmingai prognozuoti ateities kainas, galima gauti didelę finansinę grąžą. Daugelis jau atliktų eksperimentų pagrindinį dėmesį skiria sentimento analizavimui, nes žmonių nuomonė, kuria jie dalinasi socialiniuose tinkluose, gali turėti įtakos kriptovaliutų kainoms.

Toliau pateikta panašių ir reikšmingų darbų, kurie gali būti naudingi ir šiam darbui, apžvalga.

Pirmame darbe [SN18] naudojami LSTM ir ARIMA modeliai siekiant nuspėti tokių finansinių indeksų kaip Nikkei 225 ir NASDAQ ateities kainas, kai vieninteliais duomenys buvo praeities kainos(mėnesiniai duomenys). Pagal RMSE rodiklį, gauta kad LSTM modelis pasirodė daug geriau nei ARIMA, buvo fiksotas vidutiniškai -87% RMSE sumažėjimas.

Kito panašaus darbo [ABR17] autoriai turėdami mėnesinius BTC duomenis iš 2013 iki 2017 metų pritaikė ARIMA modelį ir bandė nuspėti BTC ateities kainas. Buvo nustatyta, kad BTC eilutė nėra stacionari, taip pat, rodiklis MAPE buvo lygus 5.36%. Autoriai pastabosė taip pat nurodė kad modelio panaudojimui reikalinga detalesnė paklaidų analizė, nes BTC kainų duomenys yra labai nepastovūs.

Trečiame darbe [SJOL20] naudojami įrašai iš Twitter platformos su žyme, kad įraše yra informacija apie BTC kriptovaliutą. Taip pat minutiniai BTC kainos duomenys. Kadangi socialinių medijų įrašuose nėra jokių reikalavimų, jie dažnai būna nerilišios kalbos, turi akronimų ar sutrumpinimų, emotikonų (angl. emoji). Autoriai sprendė šią problemą pašalindami iš įrašų visus simbolius, kurie nebuvo tekstu, suvienodino raidžių didumą, klasifikavo ir pašalino emotikonus. Sentimentui išgauti buvo naudojamas VADER<sup>1</sup> įrankis. Prognozavimui pasirinktas Atsitiktinių miškų regresijos (angl. Random Forest Regression) modelis. Vertinimo rodiklis vidutinė paklaida siekė 37.52%. Autoriai pastebi, kad nors ir gauta per didelę vidutinę paklaida sėkmingai prognozei, tačiau rasta stipri koreliacija tarp BTC kainos procentinio pokyčio ir Twitter įrašų sentimento.

Kito darbo [AEK<sup>+</sup>21] autorių tikslas buvo nuspėti BTC kriptovaliutos kainos svyravimus (dispersiją). Surinkti 15 minučių dažnumo BTC kainos duomenys (14'000 įrašų) ir įrašai iš Twitter platformos apie BTC kriptovaliutą (30'000'000 įrašų). Kad informacija būtu lengviau apdorojama, iš teksto buvo išgauti tokie požymiai kaip įrašo tipas, jautrios kalbos statu-

---

<sup>1</sup> Hutto, C.J. and Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

sas, vartotojas, sentimentui gauti naudotas ankstesnėje pastraipoje minėtas VADER<sup>1</sup> modelis. Išbandyti LSTM, GRU, TCN, GARCH prognozavimo modeliai. Įvertinimui ir palyginimui naudoti MAPE, MAE, RMSE, MSLE. Gauta, kad geriausi rezultatai ( $MAPE = 0.2$ ) buvo gauti su TCN modeliu naudojant informaciją apie vartotoją, kuris paskelbė įrašą, kas yra paaiškinama, nes daugelis spekuliuojančių žmonių sekā ir kopijuojat tai, ką daro už juos sėkmingesni vartotojai.

Šiame darbe [AHNI18] dėmesys skirtas BTC ir ETH kriptovaliutų kainų prognozei pasitelkiant Twitter įrašais. Surinkti Twitter įrašų duomenys apie BTC ir ETH kriptovaliutas, Google Trends duomenys, dieninis Twitter bendras įrašų skaičius, BTC ir ETH dieninės kainos. Sentimentui gauti pa- naudotas VADER<sup>1</sup> modelis, sukurti tokie požymiai kaip SVI iš Google Trends duomenų, taip pat paskaičiuoti kainų pokyčiai tarp dienų. Pritaikius tiesinės regresijos modelį nebuvo gauti geri rezultatai, autoriaus teigimu taip gali būti dėl to, kad ryšys tarp kainų ir sukurtų požymių nėra tiesinis. Rezultatuose taip pat pastebėta, kad Google Trends duomenys stipriai koreliuoja su BTC ir ETH kainomis.

### 3 Pirminė duomenų analizė

Kriptovaliutų ir kitų aktyvų (angl. assets) kainų duomenis gavome iš švedų internetinio banko svetainės Dukascopy<sup>2</sup>. Gavome tokį aktyvų kaip XRP, BTC, SPY, VXX, XAU kainas dieniniai, valandiniai ir minutiniai intervalais, taip pat duomenys buvo prieinami tik arba iš pirkėjo (angl. bid) arba pardavėjo (angl. ask) pusės, tad iš viso pradžioje turėjome 30 failų.

Duomenys buvo nuo 2022-01-07 00:01:00 iki 2022-02-28 23:52:00.

Dieniniai duomenys turėjo 55 eilutes, valandiniai - 1271, minutiniai - 76312. Po duomenų valymo turėjome 3 duomenų rinkinius kuriame kiekviename buvo 5 aktyvų kainos.

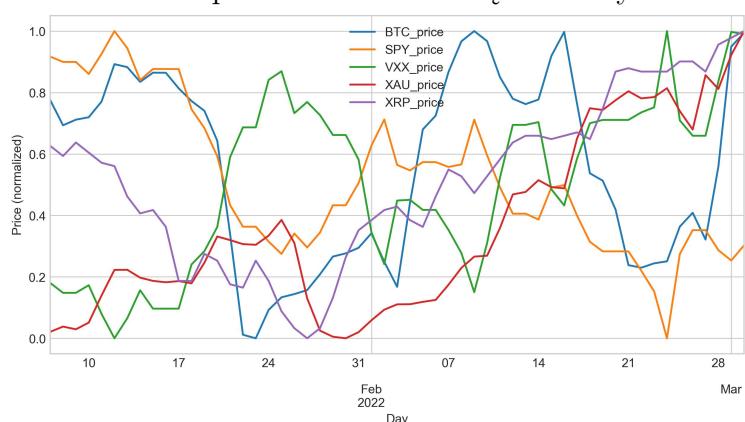
Prieš bražiant kainų grafikus, duomenys buvo normalizuoti, nes aktyvų skaičius stipriai skyrėsi. Po normalizavimo visos kainos įgyjo reikšmes intervale [0;1].

Dieninių aktyvų kainų grafike (žr. 1 pav.) nesimato tendencijų arba priklausomybių tarp XRP kainų ir kitų finansinių aktyvų kainų. Matome, jog sausio mėnesį XRP kaina greitai mažėjo, kol sausio pabaigoje nepasiekė savo minimumo. Vasario mėnesį šios kriptovaliutos kaina, kaip ir XAU, VXX kainos augo.

Valandiniuose aktyvų kainų duomenyse (žr. 2 pav.) ryškiai matosi daug kainų šuolių, sunkiau įžiūrimos ilgalaikės tendencijos. XRP kainos valandiniuose intervaluose keitėsi daug dažniau.

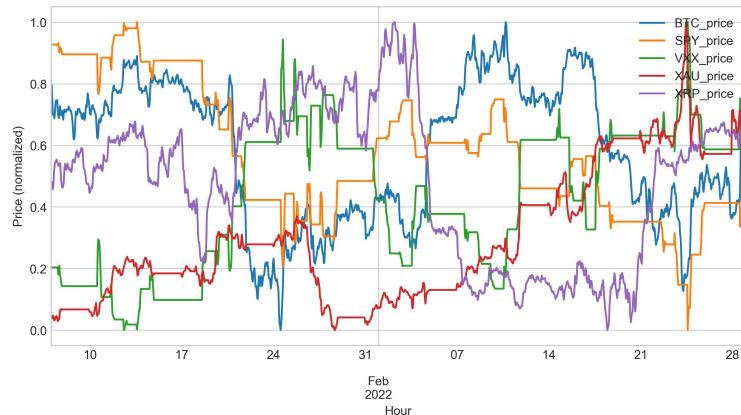
Minutiniuose duomenyse (žr. 3 pav.), kaip ir valandiniuose, kainos kito labai greitai. Nei viename iš trijų - minutinių, valandinių, dieninių duomenų - grafikų nepastebimos stiprios koreliacijos tarp XRP ir kitų finansinių aktyvų kainų, todėl prognozuoti XRP kainą atsižvelgiant į kitų aktyvų kainas nėra prasmės.

1 pav.: Dieniniai kainų duomenys

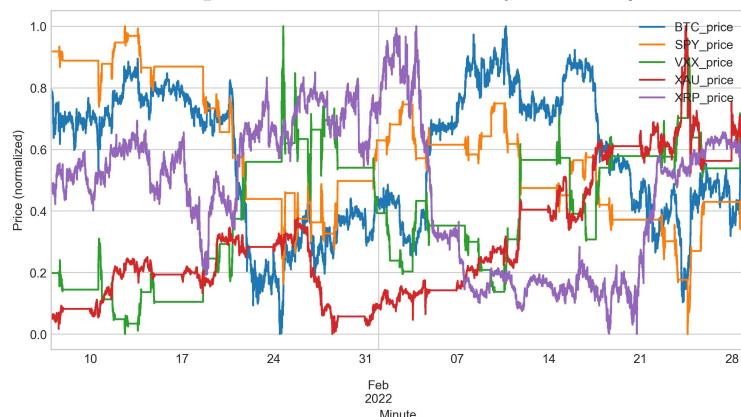


<sup>2</sup> Dukascopy Bank - <https://www.dukascopy.com>

2 pav.: Valandiniai kainų duomenys

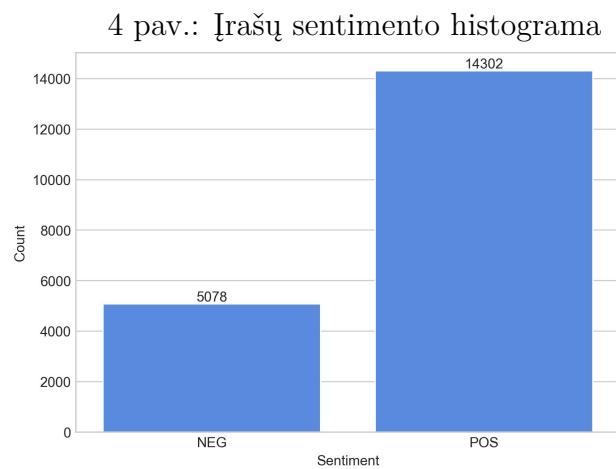


3 pav.: Minutiniai kainų duomenys



Kitas duomenų rinkinys – tai Twitter įrašų duomenys apie XRP. Jame yra 2 stulpeliai: įrašo tekstas ir įrašo paskelbimo laikas minučių tikslumu. Pastebėta, kad įrašų tekste dažnai yra nenaudingos informacijos tokios kaip nuorodos, kitų valiutų ar vartotojų minėjimai, tad ji buvo pašalinta.

Įrašo sentimentui išgauti buvo naudojamas RoBERTa modelis pritaikytas finansinės temos įrašams iš Twitter. Iš viso turima 19378 Twitter įrašų apie XRP. Stulpelinė diagrama (žr. 4 pav.) parodo kiek iš jų yra pozityvūs bei kiek neigiami. Diagramoje NEG reiškia neigiamą sentimentą, POS - pozityvų. Iš grafiko aiškiai matyti, kad teigiamų įrašų skaičius (14302) stipriai lenkia neigiamų įrašų skaičių (5078), viena iš to priežasčių yra kad teigiamos žinutes skatina kainos kilimą, kas ir yra vienas iš pagrindinių investuotojų tikslų.



Tuomet buvo nuspręsta pasižiūrėti kaip atrodo bendras dienos įrašų skaičius, nuspalvintas pagal sentimento dominavimą tą dieną.

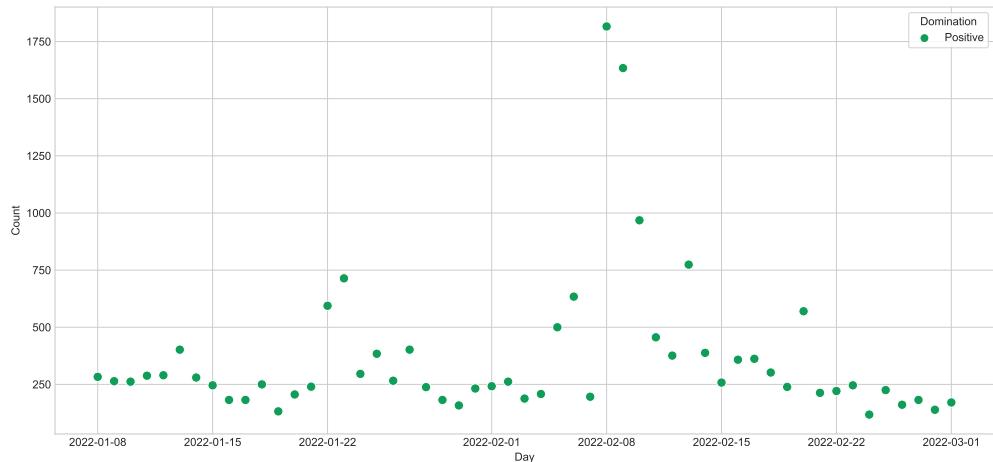
Dieniniuose duomenys (žr. 5 pav.) nebuvo tokios dienos, kad dominuotų neigiami įrašai, manome tai yra dėl to, nes diena yra ilgas laiko tarpas kriptovaliutų kainų judėjimui ir neigiami komentarai yra užgožiamai labai dideliu kiekiu teigiamų įrašų.

Valandiniuose duomenyse (žr. 6 pav.) galime pastebėti, kad buvo labai nedaug valandų su dominuojančiu neigiamu sentimentu, tačiau tokios valandos pasitaikydavo pakilus bendram įrašų skaičiui.

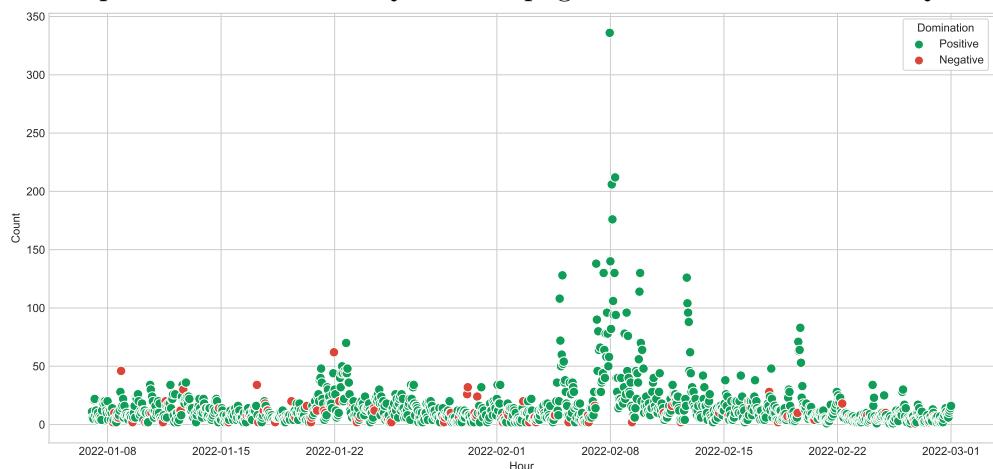
Grafike su minutiniais duomenimis (žr. 7 pav.) daugelis taškų stipriai susigrūda, tačiau paeksperimentavus su skirtingais grafikais, buvo padaryta išvada, kad taškų grafikas geriausiai atvaizduoja sentimento dominavimą ir skirtinį minučių reikšmės mažiausiai persidengia. Taip pat matome, kad didėjant įrašų skaičiui, atsiranda vis daugiau minučių, kurių sentimentas yra neigiamas. Tai galima paaiškinti tuo, kad didėjant teigiamiems komentarams, didėja ir vartotojų, norinčių pasisakyti priešingai skaičius, vyksta diskusijos. Minutiniuose duomenyse minutę, kai buvo pasidalinta daugiausia įrašų apie XRP, dominavo teigiamas įrašų sentimentas.

Atrodo, kad minutiniai duomenys turėtų būti naudingiausi nes turi didžiausią sentimento variaciją tarp laiko taškų.

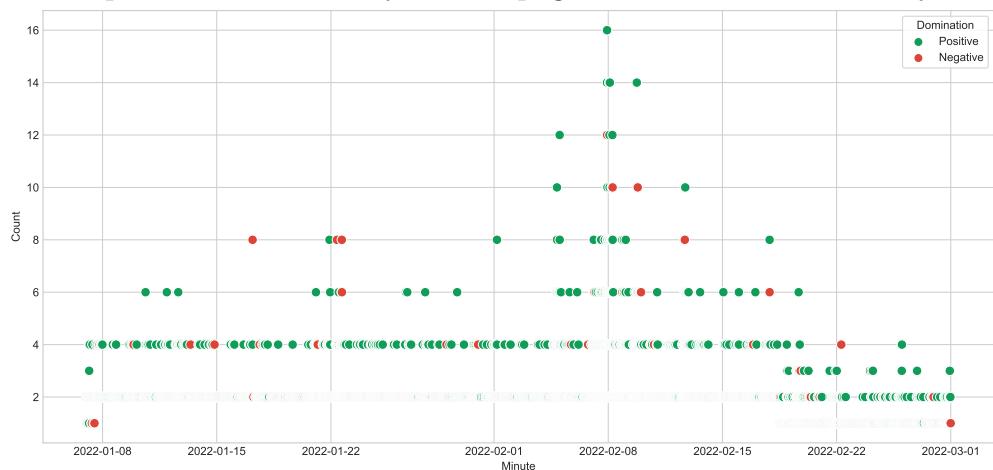
5 pav.: Dieninis irašų skaičius pagal sentimento dominavimą



6 pav.: Valandinis irašų skaičius pagal sentimento dominavimą



7 pav.: Minutinės irašų skaičius pagal sentimento dominavimą

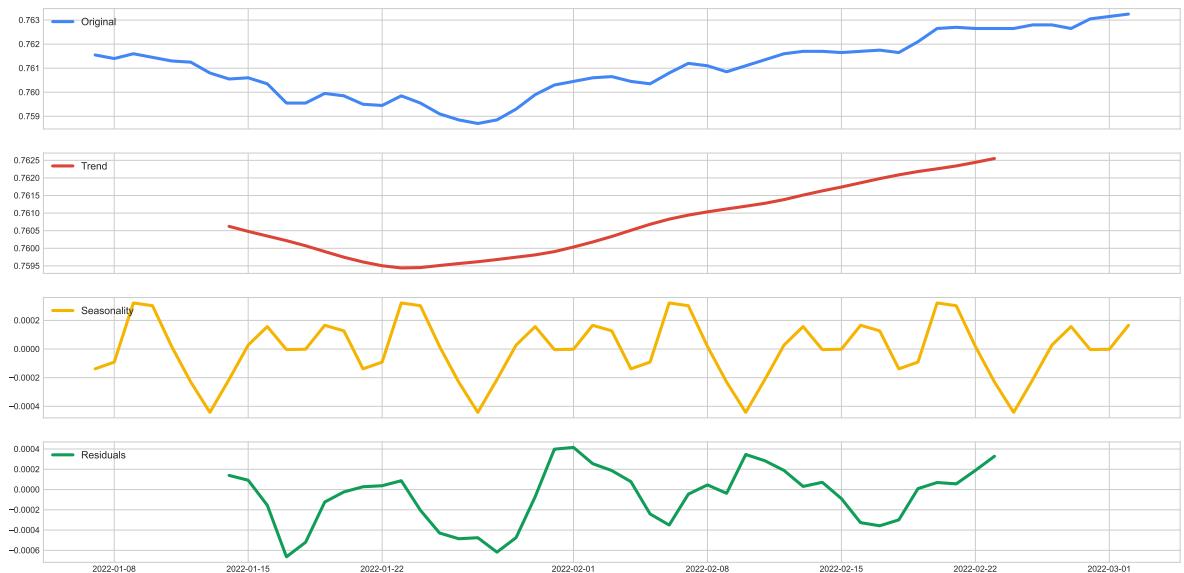


## 4 Eksperimentai

Pirmiausia buvo patikrinta kaip su turimais duomenimis veikia klasikinis ARIMA modelis.

### 4.1 Dieniniai grafikai

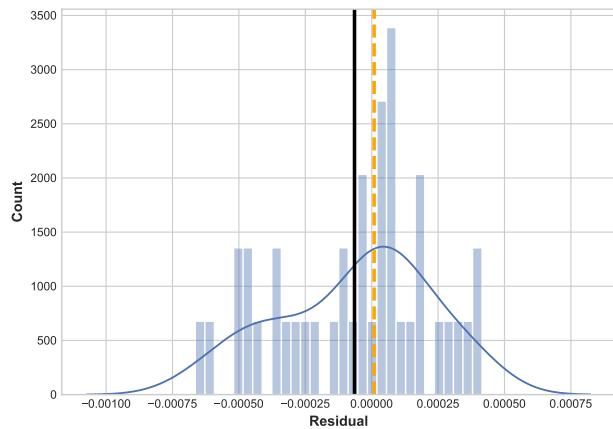
8 pav.: Dieninių duomenų dekompozicija



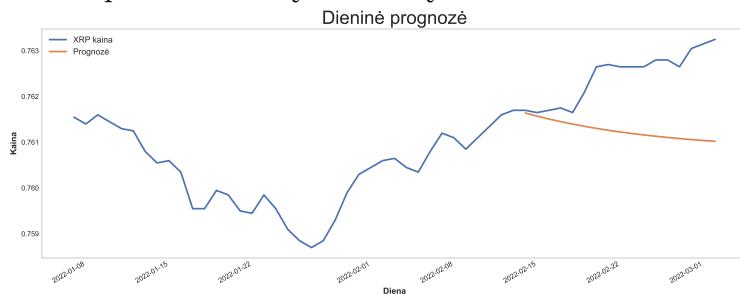
Pradėta nuo dieninių duomenų. Jie buvo išskaidyti į dedamąsias dalis - trendas, sezonišumas, liekanos (žr. 8 pav.). Dieniniuose duomenyse ryškiai matosi kainos augimo tendansas, tačiau dar per anksti teigti, kad tendansas egzistuoja, reikia patikrinti valandinius ir minutinius duomenis. Taip pat matosi pasikartojanti struktūra sezonišumo kreivėje. Atskiro dėmesio ir detalesnės analizės reikalauja liekanos. Buvo patikrinta ar dieninių duomenų liekanos normaliai pasiskirsčiusios (žr. 9 pav.). Nors kreivė gana panaši į normaliojo skirstinio kreivę, sunku pasakyti ar liekanos yra normaliai pasiskirsčiusios, kadangi duomenų yra per mažai (tik 55 dienų XRP kainos).

Išanalizavus dieninių duomenų dedamąsias, buvo bandoma prognozuoti XRP kainas (žr. 10 pav.). Grafike matosi, kad ARIMA modelis nėra tinkamas dieniniams duomenims, prognozuojamos reikšmės yra toli nuo tikrujų. Be to, matosi, jog tikroji XRP kaina auga, o ARIMA modelio spėjamos reikšmes eina žemyn.

9 pav.: Dieninių duomenų liekanų pasiskirstymas



10 pav.: Dieninių duomenų ARIMA rezultatai



## 4.2 Valandiniai grafikai

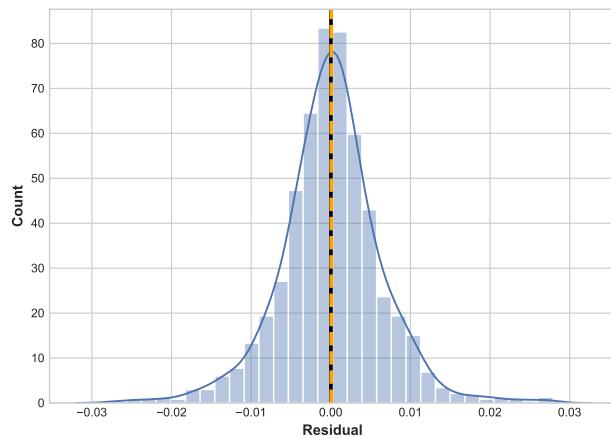
11 pav.: Valandiniai duomenų dekompozicija



Tokie pat veiksmai buvo atlikti ir su valandiniais duomenimis. Iš pra-

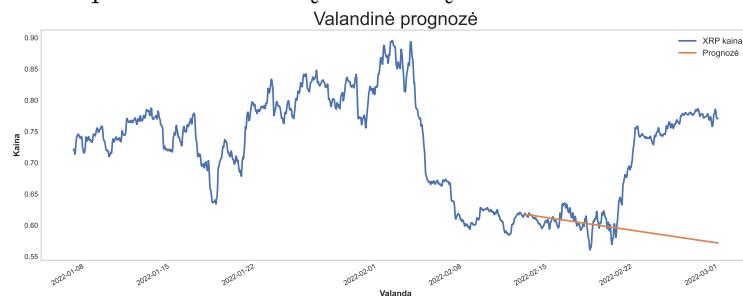
džių duomenys buvo išskaidyti į trendą, sezoniškumą, liekanas (žr. 11 pav.). Matosi, kad trendas šiek tiek pašalina triukšmą iš laiko eilutės.

12 pav.: Valandinių duomenų liekanų pasiskirstymas



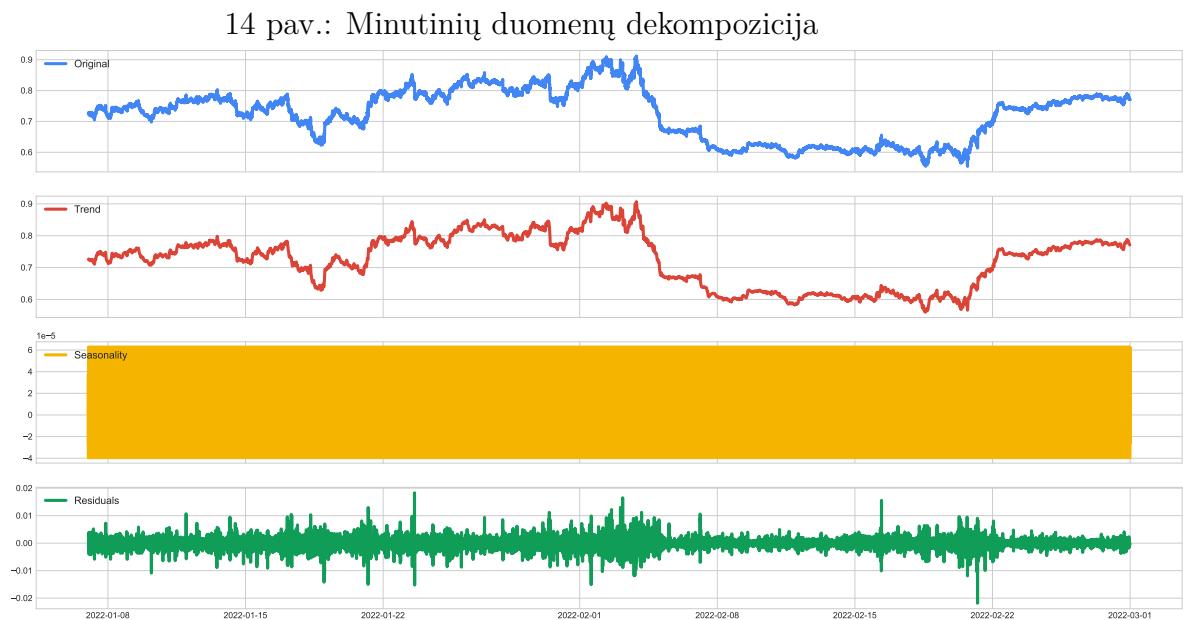
Taip pat buvo patikrinta prielaida, kad valandinių duomenų liekanos yra normaliai pasiskirsčiusios. Grafike (žr. 12 pav.) matosi, kad liekanos tikrai yra pasiskirsčiusios normaliai, simetriškai apie 0.

13 pav.: Valandinių duomenų ARIMA rezultatai



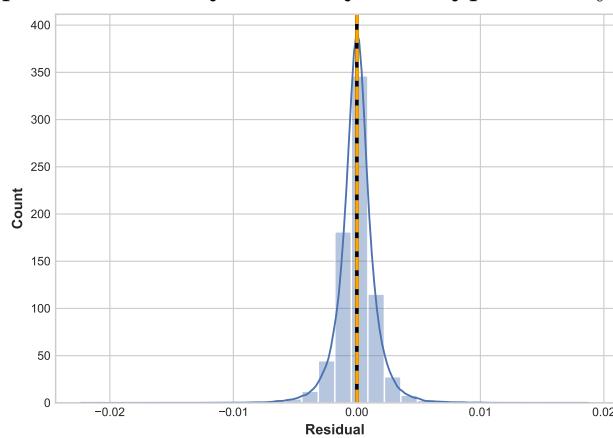
Valandiniamis duomenimis ARIMA modelis (žr. 13 pav.) prognozuoja tiesę einančią žemyn, nors kainos testavimo aibėje didelę laiko dalį kilo.

## 4.3 Minutiniai grafikai



Visi žingsniai, atliliki su dieniniais ir valandiniais duomenimis, buvo pa-kartoti su minutiniais duomenimis (žr. 14 pav.). Kadangi minutiniai duome-nys yra didelio tankio, jų tendo komponentės atvaizdavimas labai panašus į pradinę eilutę, tiesiog vietomis yra pašalintas triukšmas.

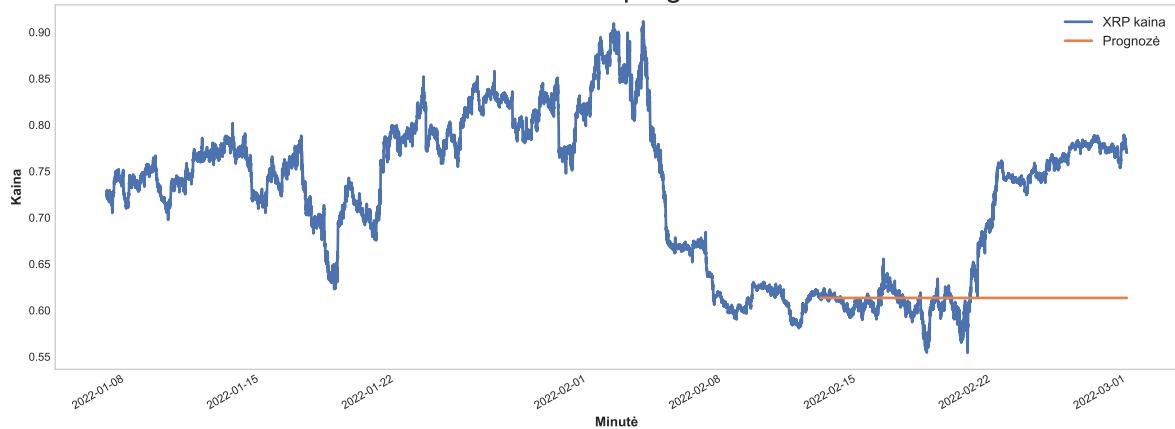
15 pav.: Minutinių duomenų liekanų pasiskirstymas



Tikrinama ar liekanos yra pasiskirsčiusios normaliai, iš grafiko (žr. 15 pav.) matosi, kad jos iš tikrujų pasiskirsčiusios pagal normalųjį dėsnį.

ARIMA modelio prognozė (žr. 16 pav.), kaip ir su valandiniais duomeni-mis, prognozuoja tiesę .

16 pav.: Minutinių duomenų ARIMA rezultatai  
Minuntinė prognozė



#### 4.4 Išvados apie ARIMA modelį

ARIMA modelis buvo pratestuotas ant dieninių, valandinių, minutinių duomenų. Su visais duomenimis modelio rezultatai buvo prasti. Taip gali būti dėl to, kad kriptovaliutų kainų duomenys yra labai nepastovūs ir neturi sezoniškumo.

## **5 Duomenų paruošimas**

Kadangi tiesinė regresija, kiti regresijos modeliai ir LSTM modelis pritaikyti tam, kad galėtų prognozuoti atsaką remiantis kitų stulpelių reikšmėmis, tad juos sukūrėme.

Pradžiai buvo suskaičiuoti BTC, SPY, VXX ir XRP procentiniai pokyčiai tarp laiko momentų, taip pat pridėti jų vėlavimai iki 4 laiko momentų atgal skaičiuojant nuo dabartinės reikšmės. Be to, pridėjome kiekvienam laiko momentui atitinkamai Twitter įrašų skaičių ir teigiamų įrašų dalį procentais. Atsakas buvo kito laiko momento procentinis XRP kainos pokytis.

Kad galėtume įvertinti kaip modeliai veikia ant nematyty duomenų, kiekviena duomenų aibė buvo suskaidyta į mokymosi, validavimo ir testavimo aibes su atitinkamais santykiais 0.56, 0.14, 0.3

Transformuoti duomenys bus naudojami tiesinei regresijai, pycaret ir LSTM skaičiavimams atlikti.

## 6 Tiesinė regresija

Kadangi kaip ir galima buvo tikėtis, ARIMA modelis su XRP kainų duomenimis suveikė gan prastai, kitas pasirinktas modelis buvo tiesinė regresija. Sudarytas modelis, kuris prognozuoja XRP kainų pokyčius skirtingais laiko intervalais.

### 6.1 Dieniniai grafikai

Buvo pradėta nuo dieninių duomenų. Nors būtent su šiuo duomenų rinkiniu gauta  $R^2$  reikšmė siekia net 90%, tačiau iš grafiko galima pastebėti, jog modelis negali tiksliai nuspėti XRP kainų procentinio pokyčio (žr. 17 pav.). Tikrieji kainų pokyčiai tarp dienų yra labai arti 0, o tiesinės regresijos modelis spėja didesnius svyravimus, nesutampančius su tikrosiomis reikšmėmis.

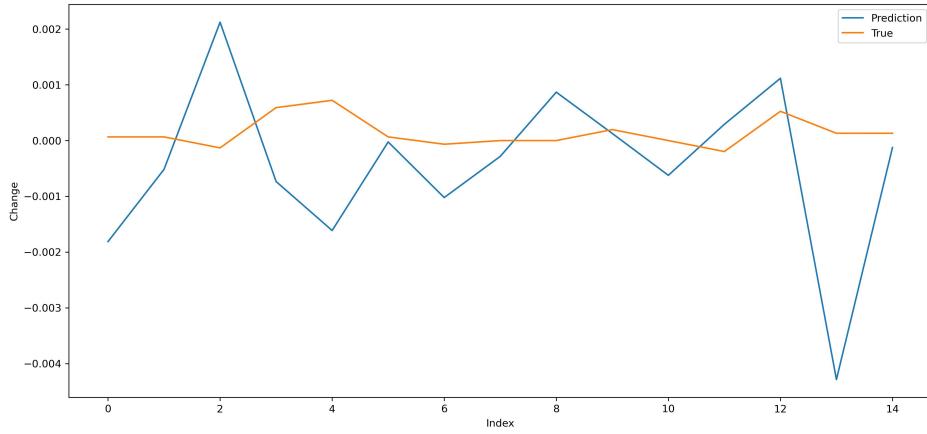
### 6.2 Valandiniai grafikai

Atlikus tiesinės regresijos skaičiavimus su valandiniais duomenimis, iš pirmo žvilgsnio gauname gan gerus rezultatus, prognozuojamos pokyčių reikšmės arti tikrujų (žr. 18 pav.). Tačiau jidėmiau įsižiūrėjus, matome, jog kai kur prognozuojami pokyčiai vėluoja nuo tikrujų per kelis laiko momentus (kelias valandas). Taip pat prie pabaigos prognozėse matosi dideli svyravimai, nors iš tikrujų jie buvo daug mažesni. Taip galėjo įvykti dėl to, nes per tas valandas Twitter buvo paskelbta daug įrašų apie XRP, taip pat pastebėjome, kad teigiamų įrašų dalis šiek tiek pakilo.

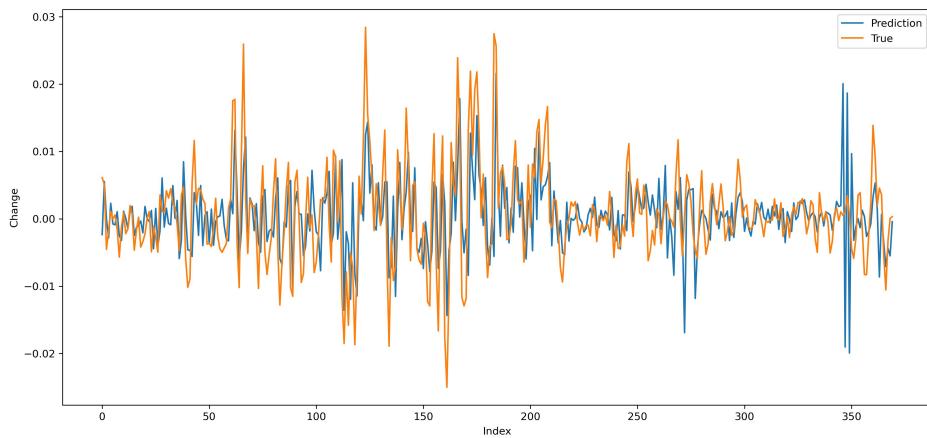
### 6.3 Minutiniai grafikai

Tiesinė regresija su minutiniiais duomenimis spėja šiek tiek mažesnius svyravimus negu tikruose duomenyse (žr. 19 pav.). Bet kadangi minutinių duomenų testavimo aibėje turėjome virš 2500 eilučių, grafike tikrosios ir prognozuojamos pokyčių reikšmės iš dalies persidengia. Todėl kai kuriose laiko atkarpose sunku įžiūrėti prognozuojamus pokyčius. Vizualiai galime pastebėti, kad modelis supranta kur yra stipresni kainų judėjimai.

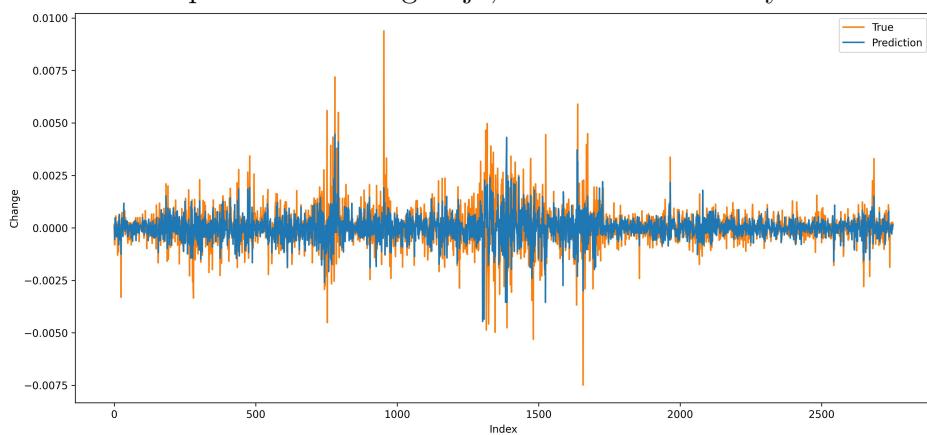
17 pav.: Tiesinė regresija, dieniniai duomenys



18 pav.: Tiesinė regresija, valandiniai duomenys



19 pav.: Tiesinė regresija, minutiniai duomenys



## 7 Pycaret biblioteka

Nagrinėjant naujausius kriptovaliutų prognozės įrankius buvo rastas Pycaret [Ali20] paketas. Paketas bando automatizuoti modelio parinkimą pritaikydamas visus tame esančius įrankius ir lygina gautas paklaidas, taip gaunant modelį, kurio paklaida yra mažiausia duotiems duomenims. Taip pat pakeite yra lengvai prieinami hiperparametru optimizavimas, modelio diagnostė ir aiškinamieji grafikai. Pycaret paketas gali spręsti 4 tipų uždavinius: klasifikavimo, regresijos, klasterizavimo, anomalijų radimo, tačiau šiame darbe jis bus naudojamas tik regresijos uždaviniams.

### 7.1 Dieniniai grafikai

Ekspериментуojant su Pycaret biblioteka buvo ieškoma geriausio modelio dieniniams duomenims. Dieninių duomenų atveju geriausiu modeliu išrinktas Extra Trees Regressor modelis (žr. 20 pav.), tačiau verta paminėti, kad jis labai nedaug skyrési nuo kitų kelių geriausių modelių.

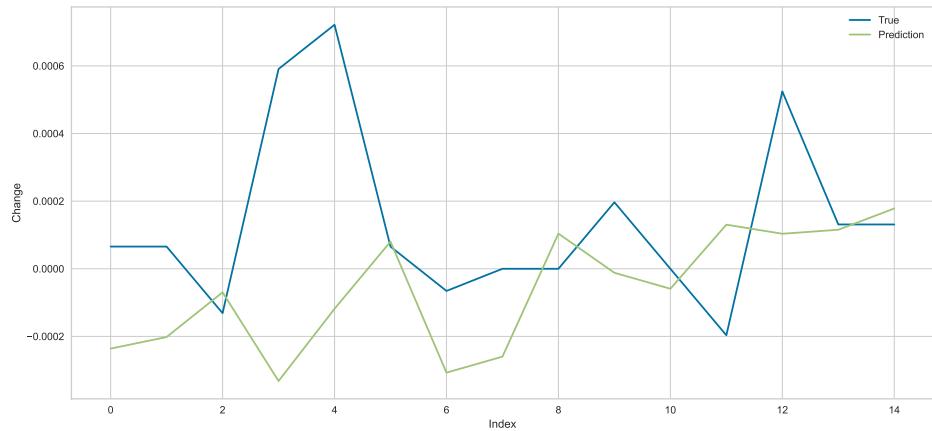
20 pav.: Modelių palyginimas, dieniniai duomenys

Model		MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	0.0003	0.0000	0.0004	-1.3418	0.0003	1.0842	0.0180
par	Passive Aggressive Regressor	0.0004	0.0000	0.0004	-1.7000	0.0004	1.0000	0.0020
knn	K Neighbors Regressor	0.0004	0.0000	0.0004	-1.7369	0.0003	1.1139	0.0020
gbr	Gradient Boosting Regressor	0.0003	0.0000	0.0004	-1.8789	0.0003	1.0897	0.0050
en	Elastic Net	0.0004	0.0000	0.0004	-2.1439	0.0004	1.0296	0.0020
lasso	Lasso Regression	0.0004	0.0000	0.0004	-2.1439	0.0004	1.0296	0.0020
dummy	Dummy Regressor	0.0004	0.0000	0.0004	-2.1439	0.0004	1.0296	0.0020
lightgbm	Light Gradient Boosting Machine	0.0004	0.0000	0.0004	-2.1439	0.0004	1.0296	0.1830
llar	Lasso Least Angle Regression	0.0004	0.0000	0.0004	-2.1439	0.0004	1.0296	0.0020
rf	Random Forest Regressor	0.0004	0.0000	0.0005	-2.3399	0.0003	1.1398	0.0220
ridge	Ridge Regression	0.0004	0.0000	0.0004	-2.4557	0.0004	1.0069	0.0020
br	Bayesian Ridge	0.0004	0.0000	0.0005	-2.5385	0.0004	1.0161	0.0020
ada	AdaBoost Regressor	0.0004	0.0000	0.0005	-2.8528	0.0003	1.3239	0.0090
huber	Huber Regressor	0.0004	0.0000	0.0005	-3.8121	0.0003	1.3698	0.0030
omp	Orthogonal Matching Pursuit	0.0004	0.0000	0.0005	-3.8648	0.0003	1.4204	0.0030
dt	Decision Tree Regressor	0.0006	0.0000	0.0006	-5.4507	0.0003	2.0761	0.0020
lr	Linear Regression	0.0041	0.0001	0.0046	-512.7521	0.0042	17.0219	0.1880
lar	Least Angle Regression	1079477.7028	12749707570610.1348	1419970.6936	-174090806669842186240.0000	9.1975	5187141910.9416	0.0030

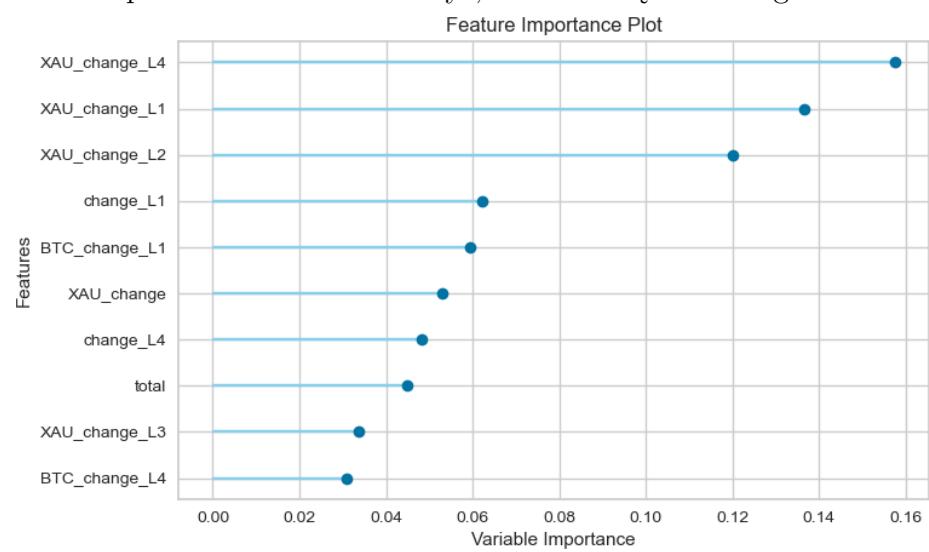
Dieninius XRP kainų pokyčius Extra Trees Regressor modelis prognozuoją gan neblogai, modelis prastai nustato pokyčių judėjimo kryptį, taip pat neatspėja judėjimo stiprumo (žr. 21 pav.). Grafike matosi, jog prognozuojamus pokyčių reikšmės yra šiek tiek mažesnės nei tikrieji pokyčiai.

Taip pat svarbu buvo sužinoti kokie regresoriai daro didžiausią įtaką būsiems XRP kainų pokyčiams. Grafike pavaizdavus 10 skirtinį kovariančių reikšmingumus, matome, jog didžiausią įtaką daro aukso indekso svyravimai (žr. 22 pav.). Reikšmingiausia kovariante išrinktas aukso pokytis prieš 4 laiko momentus (4 dienas). Be aukso kainų pokyčių svarbus ir XRP praeito laiko momento pokytis.

21 pav.: Dieninių XRP kainų pokyčių prognozavimas



22 pav.: Dieniniai duomenys, kovariančių reikšmingumas



## 7.2 Valandiniai grafikai

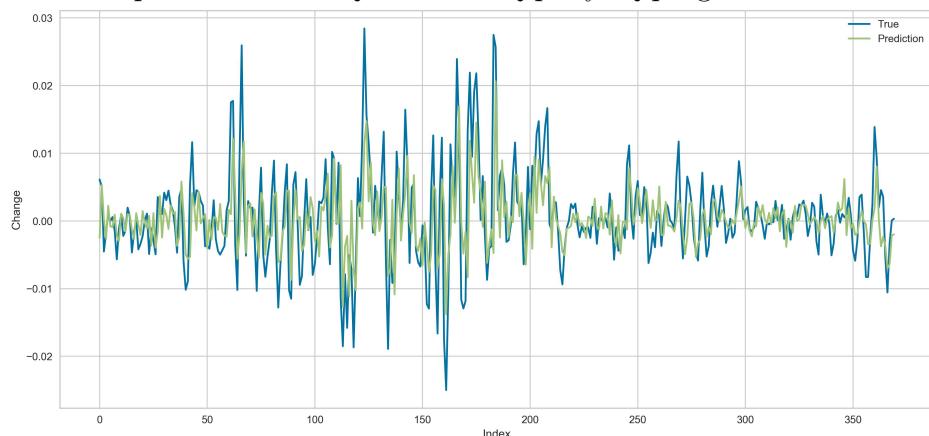
Valandiniam duomenų rinkiniui geriausiu modeliu buvo išrinkta Bajeso Ridge regresija (žr. 23 pav.).

23 pav.: Modelių palyginimas, valandiniai duomenys

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
<b>br</b>	Bayesian Ridge	0.0045	0.0000	0.0061	0.3242	0.0051	1.8621	0.0020
<b>lr</b>	Linear Regression	0.0045	0.0000	0.0061	0.3183	0.0050	1.8898	0.0030
<b>lar</b>	Least Angle Regression	0.0046	0.0000	0.0062	0.3087	0.0050	1.9098	0.0030
<b>omp</b>	Orthogonal Matching Pursuit	0.0047	0.0000	0.0063	0.2846	0.0053	1.8279	0.0030
<b>huber</b>	Huber Regressor	0.0047	0.0000	0.0063	0.2843	0.0053	1.8851	0.0110
<b>et</b>	Extra Trees Regressor	0.0050	0.0000	0.0066	0.2021	0.0056	2.2351	0.0510
<b>gbr</b>	Gradient Boosting Regressor	0.0050	0.0000	0.0067	0.1821	0.0056	2.4110	0.0470
<b>rf</b>	Random Forest Regressor	0.0050	0.0000	0.0068	0.1635	0.0056	2.3366	0.0800
<b>ada</b>	AdaBoost Regressor	0.0051	0.0000	0.0069	0.1381	0.0057	2.1784	0.0210
<b>lightgbm</b>	Light Gradient Boosting Machine	0.0052	0.0000	0.0068	0.1270	0.0054	2.5896	0.2640
<b>par</b>	Passive Aggressive Regressor	0.0055	0.0001	0.0076	-0.0302	0.0075	1.0000	0.0030
<b>lasso</b>	Lasso Regression	0.0055	0.0001	0.0076	-0.0385	0.0074	1.0219	0.0020
<b>en</b>	Elastic Net	0.0055	0.0001	0.0076	-0.0385	0.0074	1.0219	0.0030
<b>llar</b>	Lasso Least Angle Regression	0.0055	0.0001	0.0076	-0.0385	0.0074	1.0219	0.0020
<b>dummy</b>	Dummy Regressor	0.0055	0.0001	0.0076	-0.0385	0.0074	1.0219	0.0020
<b>ridge</b>	Ridge Regression	0.0055	0.0001	0.0076	-0.0423	0.0072	1.1486	0.0020
<b>knn</b>	K Neighbors Regressor	0.0059	0.0001	0.0078	-0.1188	0.0063	2.0158	0.0030
<b>dt</b>	Decision Tree Regressor	0.0073	0.0001	0.0099	-0.9003	0.0075	3.7548	0.0050

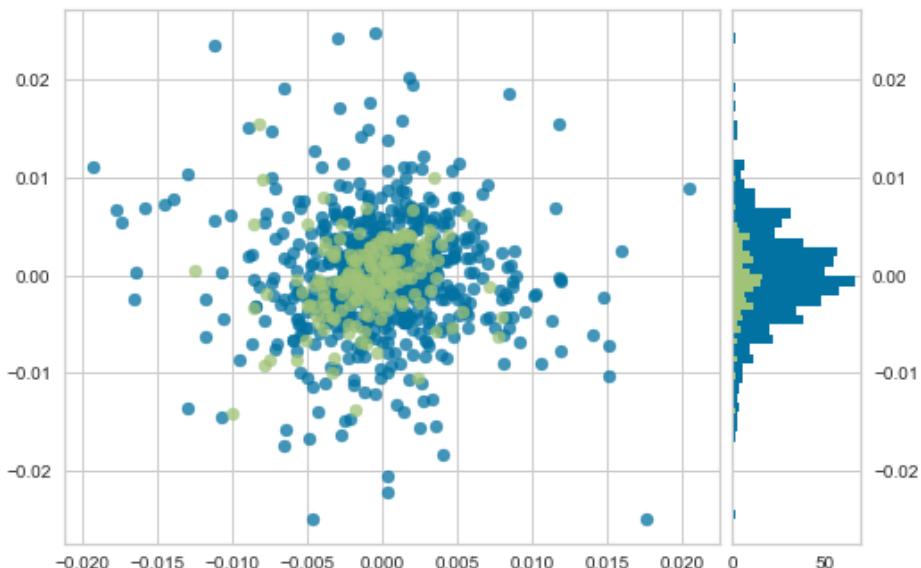
Su valandiniais duomenimis kainų pokyčių tendencijos buvo prognozuojamos gerai, modelis teisingai atspėdavo kainų didėjimą/mažėjimą (žr. 24 pav.). Bet svarbu pastebėti, kad modelis neteisingai prognozuoja pokyčių stiprumą, t.y. prognozuojami pokyčiai beveik visur mažesni už tikruosius.

24 pav.: Valandinių XRP kainų pokyčių prognozavimas



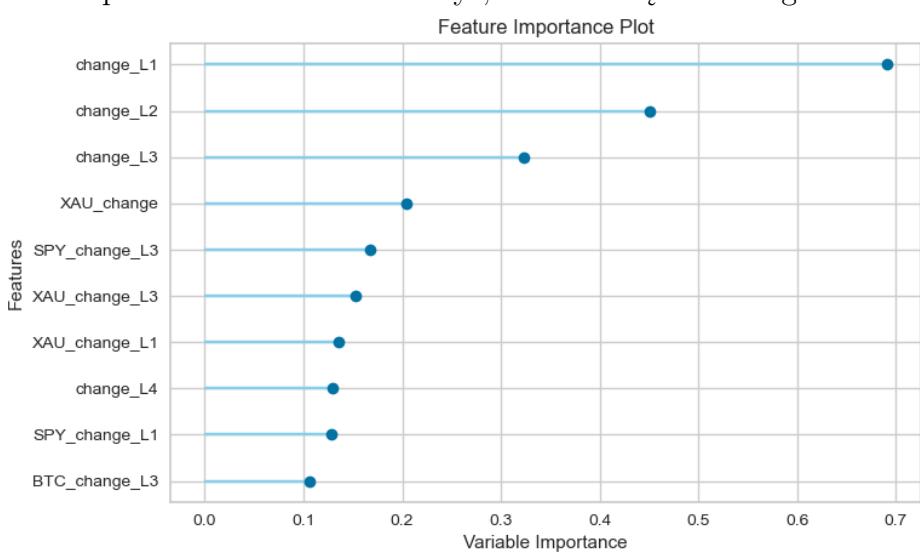
Taip pat buvo patikrinta ar šio modelio prognozių paklaidos normaliai pasiskirsčiusios. Grafike matome, kad iš tikrujų XRP kainų pokyčių prognozių paklaidos pasiskirsčiusios normaliai, vidurkis apytiksliai 0 ir testavimo aibės paklaidos (žalia) neišlipa iš mokymosi aibės paklaidų (mėlyna) (žr. 25 pav.).

25 pav.: Valandinių XRP kainų pokyčių prognozių paklaidos



Valandinių duomenų rinkinyje tarp visų kovariančių reikšmingiausios buvo XRP kainų pokyčiai su skirtingais vėlavimais (žr. 26 pav.). Ypač išsiskiria pokytis prieš 1 valandą. Taip pat ir aukso pokytis buvo reikšmingas.

26 pav.: Valandiniai duomenys, kovariančių reikšmingumas



### 7.3 Minutiniai grafikai

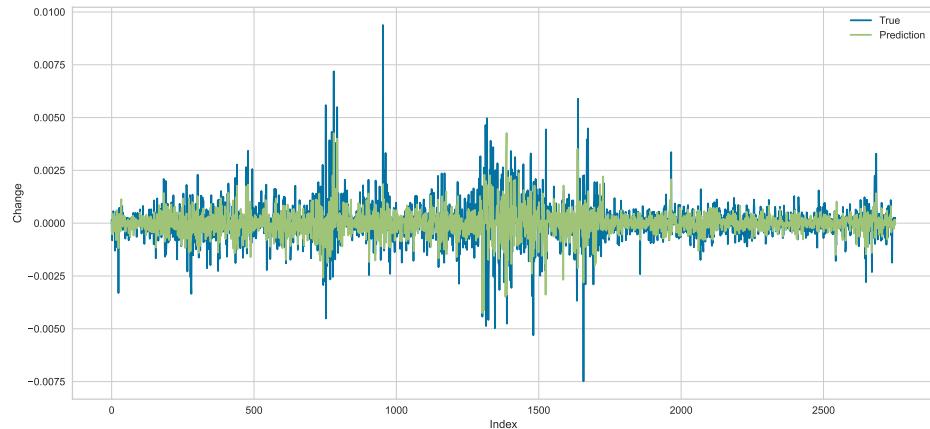
Minutiniams duomenims geriausiu modeliu išrinkta Mažiausių kampų regresija (angl. Least Angle Regression) (žr. 27 pav.).

27 pav.: Modelių palyginimas, minutiniai duomenys

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
<b>lar</b>	Least Angle Regression	0.0005	0.0000	0.0008	0.3783	0.0006	2737860156.1035	0.0030
<b>br</b>	Bayesian Ridge	0.0005	0.0000	0.0008	0.3757	0.0006	2816451593.0491	0.0030
<b>lr</b>	Linear Regression	0.0005	0.0000	0.0008	0.3748	0.0006	2852151142.7661	0.0040
<b>huber</b>	Huber Regressor	0.0005	0.0000	0.0008	0.3448	0.0007	2421087202.9441	0.0220
<b>omp</b>	Orthogonal Matching Pursuit	0.0006	0.0000	0.0008	0.3447	0.0007	2338003826.9622	0.0030
<b>gbr</b>	Gradient Boosting Regressor	0.0006	0.0000	0.0008	0.3409	0.0007	1940306434.8933	0.3210
<b>rf</b>	Random Forest Regressor	0.0006	0.0000	0.0008	0.3082	0.0007	1428086357.8729	0.7030
<b>et</b>	Extra Trees Regressor	0.0006	0.0000	0.0008	0.2984	0.0007	2048434734.6712	0.2950
<b>lightgbm</b>	Light Gradient Boosting Machine	0.0006	0.0000	0.0008	0.2834	0.0007	2105768016.5522	0.2130
<b>ada</b>	AdaBoost Regressor	0.0006	0.0000	0.0009	0.2141	0.0008	2388813237.2966	0.0850
<b>knn</b>	K Neighbors Regressor	0.0006	0.0000	0.0009	0.0964	0.0008	1734310775.6384	0.0140
<b>par</b>	Passive Aggressive Regressor	0.0007	0.0000	0.0010	-0.0044	0.0010	1.0000	0.0050
<b>ridge</b>	Ridge Regression	0.0007	0.0000	0.0010	-0.0048	0.0010	260036777.9084	0.0020
<b>lasso</b>	Lasso Regression	0.0007	0.0000	0.0010	-0.0050	0.0010	209217750.4992	0.0030
<b>en</b>	Elastic Net	0.0007	0.0000	0.0010	-0.0050	0.0010	209217750.4992	0.0030
<b>llar</b>	Lasso Least Angle Regression	0.0007	0.0000	0.0010	-0.0050	0.0010	209217747.4170	0.0030
<b>dummy</b>	Dummy Regressor	0.0007	0.0000	0.0010	-0.0050	0.0010	209217750.4992	0.0040
<b>dt</b>	Decision Tree Regressor	0.0008	0.0000	0.0012	-0.5414	0.0009	8111632255.1852	0.0170

Minutinių duomenų turėjome daugiausia. Iš prognozės grafiko matosi, jog minutinius XRP pokyčius išrinktas modelis prognozuoją gerai, nuspėja kainų augimą/kritimą bei neblogai nustato ir patį pokytį (žr. 28 pav.). Tik kai kuriaiš laiko momentais matosi, kad prognozės šiek tiek vėluoja ir sutampa su jau praeities pokyčių reikšmėmis.

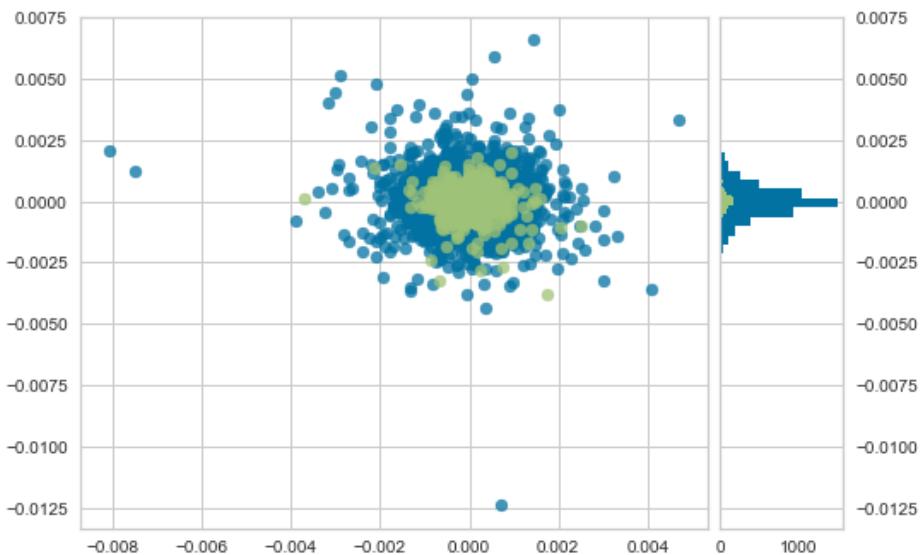
28 pav.: Minutinių XRP kainų pokyčių prognozavimas



Kaip ir su valandiniais duomenimis, gavome, kad minutinių duomenų prognozių paklaidos pasiskirsčiusios normaliai bei išsibarsčiusios apie 0, vėlgi

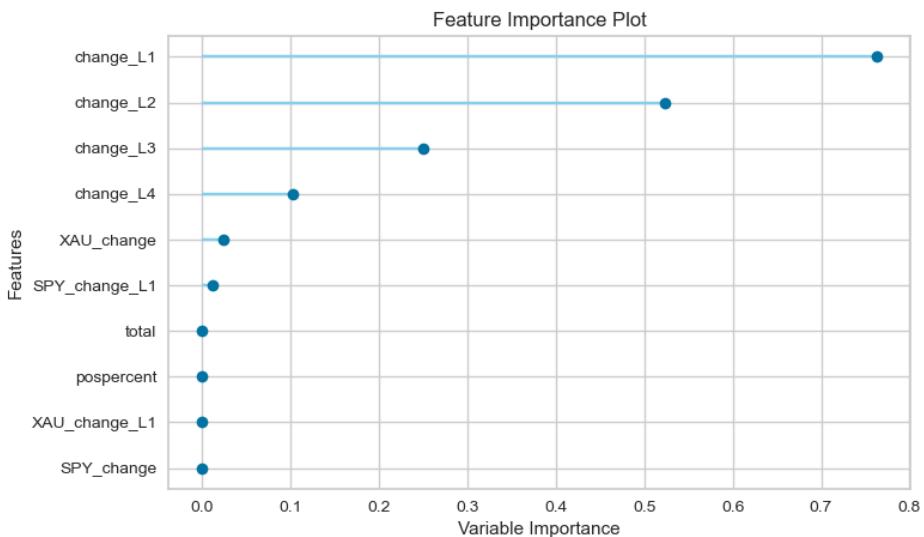
testavimo aibės paklaidų (žalia) dispersija nėra didesnė už mokymosi aibės paklaidų (mėlyna) dispersiją (žr. 29 pav.).

29 pav.: Minutinių XRP kainų pokyčių prognozių paklaidos



Su minutiniais duomenimis XRP pokyčiai ir vėl turi didžiausią įtaką XRP kainų svyravimams (žr. 30 pav.). Net 4 reikšmingiausios kovariantės yra XRP pokyčiai su skirtingais laiko momentais (minutėmis).

30 pav.: Minutiniai duomenys, kovariančių reikšmingumas



## 8 LSTM modelio sudarymas ir tyrimas

Paskutiniu modeliu buvo pasirinktas Long-Short Time Memory (LSTM) modelis. LSTM – tai rekurentinių neuroninių tinklų rūsis, kuri sugeba nustatyti priklausomybę nuo eiliškumo sekoje ir prognozuoti ateities reikšmes.

Kurdami LSTM modelį pasitelkėme ankstyvo stabdymo strategiją stabdydami modelio mokymą kai validavimo duomenų paklaida nemažėja 5 epochas. Visur buvo naudojami tokie parametrai:

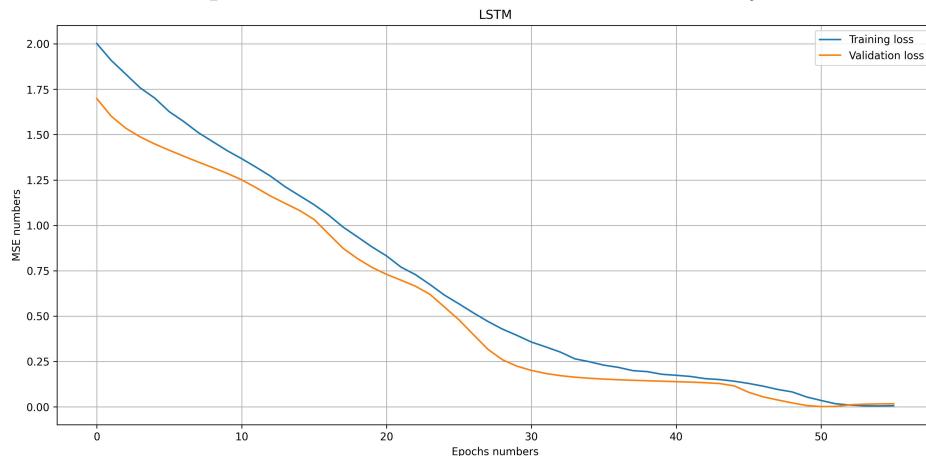
1 lentelė: LSTM modelio parametrai

Maksimalus epochų skaičius	200
Paketo dydis	64
Paklaidos funkcija	šaknis iš vidutinio nuokrypio
Optimizavimo funkcija	Adam
Išmetimo reguliariacijos procentas(angl. dropout)	24%

### 8.1 Dieniniai duomenys

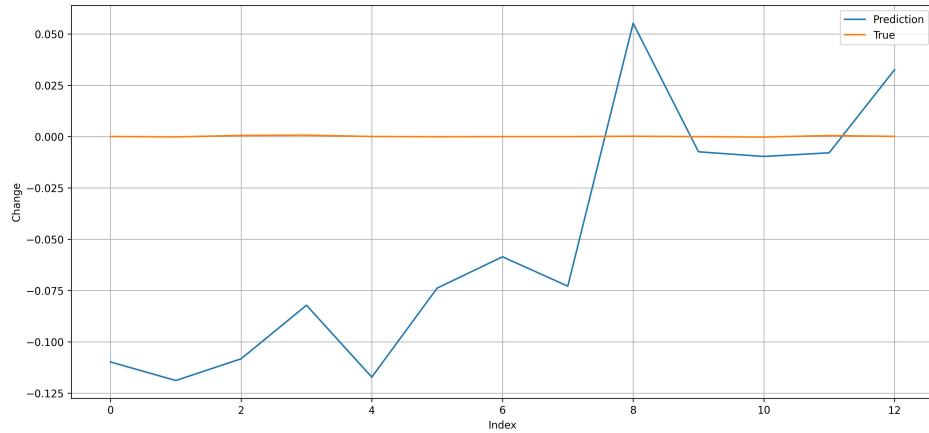
Pastebėjome kad paklaidų grafikas mokymosi ir validavimosi duomenims (žr. 31 pav.) atrodė keistai, nes validavimo paklaidos mažesnės nei mokymosi aibės, manome kad taip yra dėl to, nes turėjome labai mažai, mažiau nei 60 duomenų mokymuisi ir mažiau nei 10 duomenų validavimui.

31 pav.: LSTM liekanos, dieniniai duomenys



LSTM modelis su dieniniais duomenimis suveikė prastai (žr. 32 pav.). Modelis spėja per didelius kainų pokyčius, o tikrieji pokyčiai grafike pavaizduojami kaip tiesė, lygi 0, nors iš tikrujų tarp minučių įvyksta maži XRP kainų pokyčiai kurie prognozuojamoje skalėje nublanksta.

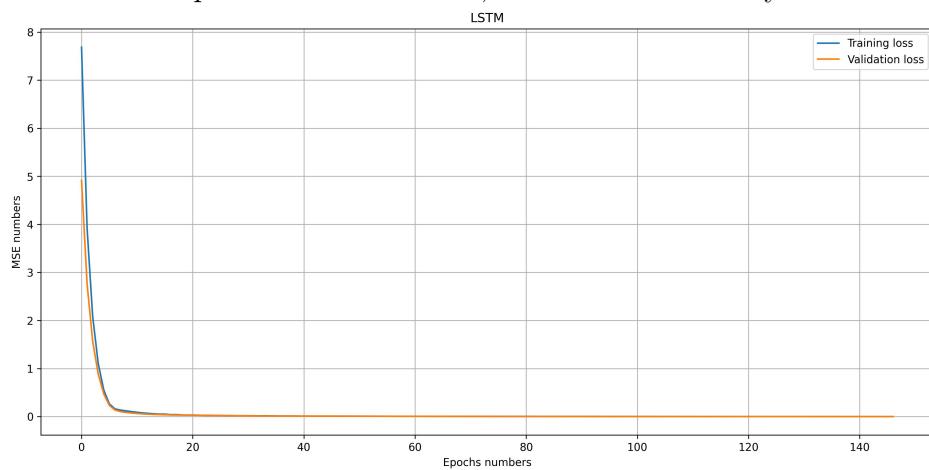
32 pav.: LSTM prognozė, dieniniai duomenys



## 8.2 Valandiniai duomenys

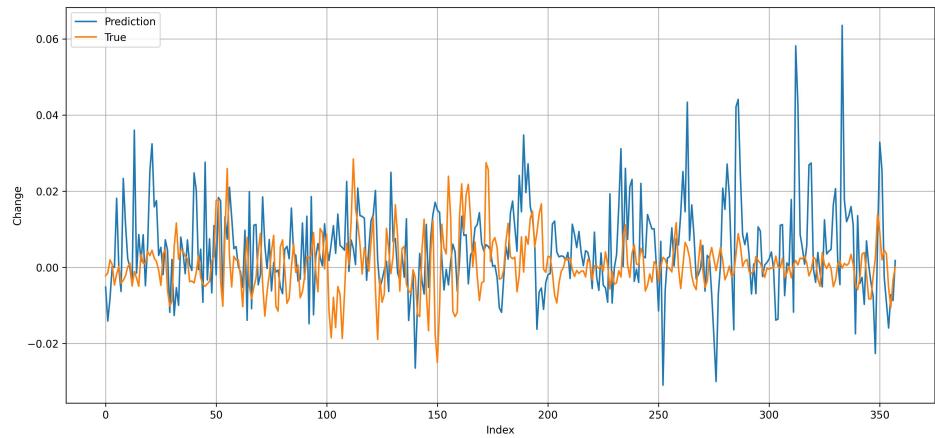
Valamdinių duomenų atveju pastebėjome kad LSTM modelis po apytiksliai 20 epochų nebegerėja reikšmingai ir toliau labai mažais pokyčiais tobulėja (žr. 33).

33 pav.: LSTM liekanos, valandiniai duomenys



Apmokius modelį su valandiniais duomenimis gaunami nelabai geri rezultatai (žr. 34 pav.). LSTM modelis bando spėti kainų pokyčius, atsižvelgiant į kovariantes, tačiau spėjamos reikšmės yra toli nuo tikrujų pokyčių ir nesutampa judėjimų kryptis. Kaip ir su dieniniu duomenų rinkiniu daug kur prognozuojamos pokyčių reikšmės yra didesnės už tikrąsias.

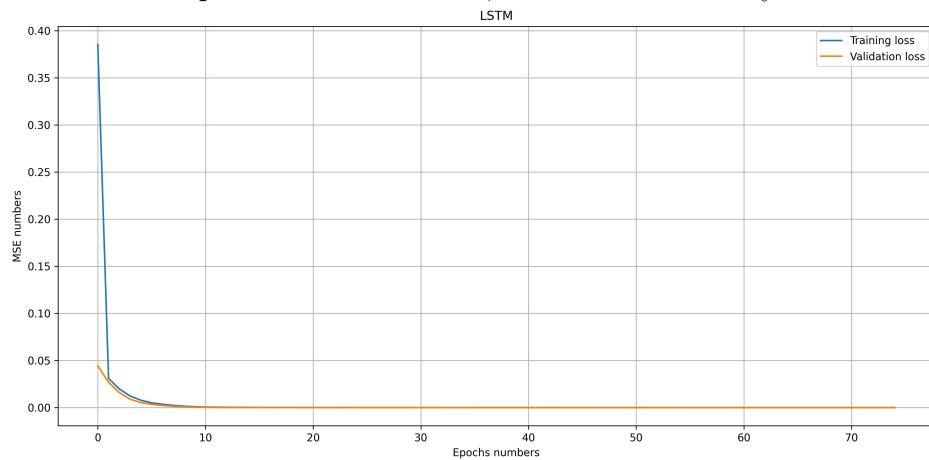
34 pav.: LSTM prognozė, valandiniai duomenys



### 8.3 Minutiniai duomenys

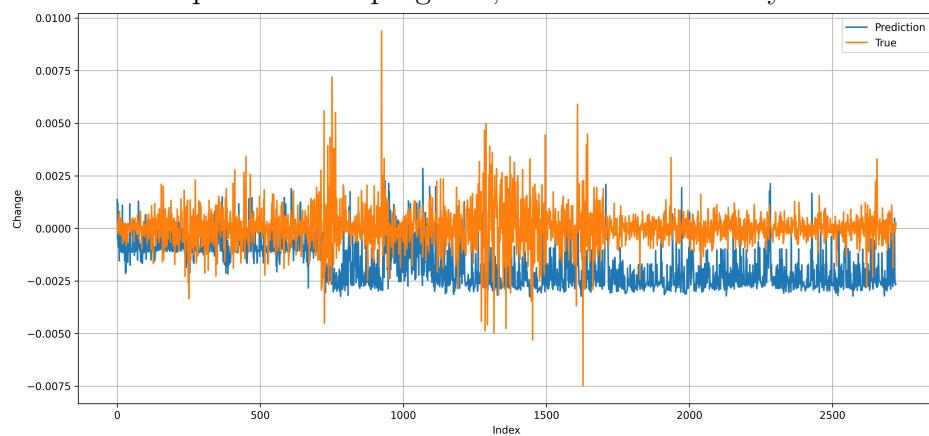
Su minutiniai duomenimis mokymasis taip pat stipriai suletojo, tik šiekart nuo 10 epochos.

35 pav.: LSTM liekanos, minutiniai duomenys



Nors buvo tikimasi, kad ant minutinių duomenų LSTM modelis suveiks geriausiai dėl didelio duomenų kiekyo, rezultatai nebuko geri. Pirmiems laiko momentams (minutėms) modelis spėja reikšmes gan panašias į tikrąsias, bet apytiksliai nuo 700-osios minutės modelis prognozuoja žymiai mažesnius pokyčius, negu yra iš tikro (žr. 36 pav.). Taip pat minutinis LSTM modelis nesugeba prognozuoti judėjimo stiprumą.

36 pav.: LSTM prognozė, minutiniai duomenys



## **8.4 Išvados apie LSTM modelį**

Nei su vienu duomenų rinkiniu LSTM modelis nesuveikė gerai.

Supratome, kad LSTM modeliui reikalingas tikrai didelis duomenų kiekis, taip pat reikia daug žinių ir supratimo kad galima būtų optimizuoti modelį specifiniam uždaviniui ir duomenis, ko manome kad mums trūko.

## 9 Išvados bei rekomendacijos

Šiame darbe buvo išbandyti keli modeliai XRP kainų pokyčiams nuspėti. Tam, kad lengviau būtų palyginti tiesinės regresijos, Pycaret atrinktus ir LSTM modelius, kiekvienam iš jų buvo paskaičiuotos MAE ir RMSE reikšmės:

2 lentelė: Dieninių duomenų modelių palyginimas

Dieniniai duomenys				
	Tiesinė regresija	Pycaret(Extra Trees Regressor)	Trees	LSTM
MAE	0.0011	<b>0.0003</b>		0.065
RMSE	0.0016	<b>0.0004</b>		0.076

3 lentelė: Valandinių duomenų modelių palyginimas

Valandiniai duomenys				
	Tiesinė regresija	Pycaret(Bayesan Ridge)	Ridge	LSTM
MAE	0.0046	<b>0.0041</b>		0.011
RMSE	0.0064	<b>0.0058</b>		0.0145

4 lentelė: Minutinių duomenų modelių palyginimas

Minutiniai duomenys				
	Tiesinė regresija	Pycaret(Least Angle Regression)	Angle	LSTM
MAE	<b>0.0005</b>	<b>0.0005</b>		0.002
RMSE	<b>0.00077</b>	0.0008		0.0023

Iš šių trijų lentelių galima pastebėti, jog MAE ir RMSE rodikliai visur yra labai maži, nei vieno modelio MAE arba RMSE neviršijo 0.02. Tačiau reikia nepamiršti, kad nors iš pirmo žvilgsnio atrodo, jog visi modeliai veikia labai gerai, jie prognozuoja XRP kainų pokyčius. XRP kainos neviršija 1 dolerio, o procentiniai pokyčiai yra dar mažesni. Todėl ir MAE bei RMSE rodikliai yra maži, renkant geriausią modelį reikia atsižvelgti ir į visų modelių prognozuotų reikšmių grafikus.

Iš grafikų matėme, jog tiesinė regresija veikė gerai, ypač su didesnio kieko

duomenų rinkiniai ir gan tiksliai spėjo Ripple kainų pokyčius. PyCaret paketas visiems duomenų rinkiniams parinko skirtingus modelius, kurie prognozavo XRP kainų pokyčius siek tiek geriau nei tiesinė regresija, manome tai yra dėl to nes PyCaret turi galimybę pritaikyti daug skirtingu regresijos modelių, kurie kiekvienas turi savo privalumų ir trūkumų. LSTM modelis su turimais duomenimis suveikė prastai, ypač tai matosi iš dieninių XRP kainų pokyčių prognozių. Taip gali būti dėl mažo duomenų kiekio.

# 10 Priedai

## 10.1 Duomenų valymas

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import numpy as np
from functools import reduce
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

instruments = [ 'BTCUSD' , 'SPY.USUSD' , 'VXX.USUSD' , 'XAUUSD' ]
def read_and_clean_data(ticker , timeframe):
    askfilename = "raw_data/" + ticker + "_Candlestick_1_" +
                 timeframe + "_ASK_01.01.2022-12.03.2022.csv"
    bidfilename = "raw_data/" + ticker + "_Candlestick_1_" +
                  timeframe + "_BID_01.01.2022-12.03.2022.csv"
    askdf = pd.read_csv(askfilename).add_prefix('ask_')
    biddf = pd.read_csv(bidfilename).add_prefix('bid_')
    df = askdf.join(biddf, lsuffix='ask_Gmt_time' , rsuffix='_
                    bid_Gmt_time')
    df[ "open" ] = ( df.ask_Open + df.bid_Open ) / 2
    df[ "close" ] = ( df.ask_Close+ df.bid_Close ) / 2
    df[ "price" ] = ( df.open + df.close ) / 2
    df = df[ [ "ask_Gmt_time" , 'price' ]]

    df = df.rename({ "ask_Gmt_time": "time" , "price": ticker [:3]
                    + "_price"}, axis = 1)
    df.time = pd.to_datetime(df.time , dayfirst =True)
    df.time = df.time.dt.tz_localize(None)

    return df

def get_full_data(timeframe):
    dfs_to_join = [read_and_clean_data(i , timeframe) for i in
                  instruments]
    df = reduce(lambda left , right: pd.merge(left , right ,on='time'
                    ) , dfs_to_join)

    return df

def clean_binance_data(filename):
    df = pd.read_csv(filename , parse_dates = [ 'date' ])
    df[ "XRP_price" ] = ( df.open + df.close ) / 2
    df = df[ [ 'date' , 'XRP_price' ]]

    return df

def plotnormed(filename , pltname , xlabel):
```

```

df = pd.read_csv(filename , parse_dates = [ 'time' ]) . set_index
( 'time' )
print( df . shape )
for i in df . columns :
    col = df[ i ]
    df[ i ] = ( col - col . min() ) / ( col . max() - col . min() )

ax = df . plot( figsize = ( 10 , 5 ) )
ax . set( xlabel=xlabel , ylabel='Price_(normalized)' )
plt . savefig( 'images/' + pltname + ".pdf" , format = "pdf" ,
    dpi = 300 , bbox_inches='tight' )
plt . savefig( 'images/' + pltname + ".jpg" , format = "jpg" ,
    dpi = 300 , bbox_inches='tight' )
plt . show()

# #### Tweets

# In [2]:


tweets = pd.read_csv( 'raw_data/xrp_tweets.csv' , parse_dates =
    True )
tweets[ "time" ] = pd.to_datetime( tweets . timestamp ,
    infer_datetime_format=True , utc = True )
tweets . time = tweets . time . dt . tz_localize( None )
tweets . time = tweets . time . dt . floor( "min" )
tweets = tweets . drop( [ 'timestamp' ] , axis = 1 )
tweets . to_csv( 'raw_data/tweets.csv' , index = False )
tweets . head()

# In [3]:


mintime = min( tweets . time )
maxtime = max( tweets . time )
mintime , maxtime

# #### Day

# In [4]:


day = get_full_data( "D" )
print( day . shape )
xrpday = clean_binance_data( "raw_data/Binance_XRPUSDT_minute.csv"
    )
day = day . join( xrpday , lsuffix='time' , rsuffix='date' )
day = day . drop( [ 'date' ] , axis = 1 )
print( day . shape )
day = day[ ( day . time >= "2022-01-07" ) & ( day . time <= "2022-03-02"
) ]. reset_index( drop = True )

```

```

print(day.shape)
day.to_csv('raw_data/day.csv', index = False)
day.head()

# In [5]:

plotnormed("raw_data/day.csv", 'day_prices_normalized', "Day")

# #### Hour

# In [6]:

hour = get_full_data("Hour")
print(hour.shape)
xrphour = clean_binance_data("raw_data/Binance_XRPUSDT_1h.csv")
hour = hour.join(xrphour, lsuffix='time', rsuffix='date')
hour = hour.drop(['date'], axis = 1)
print(hour.shape)
hour = hour[(hour.time >= mintime) & (hour.time <= maxtime)].
    reset_index(drop = True)
print(hour.shape)
hour.to_csv('raw_data/hour.csv', index = False)
hour.head()

# In [7]:

plotnormed("raw_data/hour.csv", 'hour_prices_normalized', "Hour")

# #### Minute

# In [8]:

minute = get_full_data("M")
print(minute.shape)
xrpmminute = clean_binance_data("raw_data/Binance_XRPUSDT_minute.
    csv")
minute = minute.join(xrpmminute, lsuffix='time', rsuffix='date')
minute = minute.drop(['date'], axis = 1)
print(minute.shape)
minute = minute[(minute.time >= mintime) & (minute.time <=
    maxtime)].reset_index(drop = True)
print(minute.shape)
minute.to_csv('raw_data/minute.csv', index = False)
minute.head()

```

```
# In [9]:
```

```
plotnormed("raw_data/minute.csv", 'minute_prices_normalized',  
          'Minute')
```

---

## 10.2 ARIMA modelis

```
#!/usr/bin/env python  
# coding: utf-8
```

```
# In [1]:
```

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
plt.style.use('seaborn-whitegrid')  
import seaborn as sns  
sns.set_style("whitegrid")  
sns.set_palette("deep")  
from statsmodels.graphics.gofplots import qqplot  
from scipy import stats  
from pmdarima.arima import ADFTest  
from statsmodels.tsa.seasonal import seasonal_decompose  
from darts import TimeSeries  
from darts.models import AutoARIMA  
from sklearn.metrics import mean_absolute_error as mae,  
    mean_absolute_percentage_error, mean_squared_error  
import warnings  
warnings.filterwarnings('ignore')  
GLOBAL_BLUE = "#4285F4"  
GLOBAL_RED = "#DB4437"  
GLOBAL_YELLOW = "#F4B400"  
GLOBAL_GREEN = "#0F9D58"  
  
def test_stationarity(col):  
    testresult = ADFTest(alpha=0.05).should_diff(col)  
    p, text = testresult[0], testresult[1]  
    print('p-value: ' + str(round(p, 2)))  
    print('Ar_reikia_diferencijuoti: ' + str(text))  
  
def plot_decomposition(col, period, timeframe):  
    ts_decomposition = seasonal_decompose(col, model='additive',  
                                           period=period)  
    trend_estimate = ts_decomposition.trend  
    seasonal_estimate = ts_decomposition.seasonal  
    residual_estimate = ts_decomposition.resid  
  
    fig, axes = plt.subplots(4, 1, sharex=True, sharey=False)  
    fig.set_figheight(10)
```

```

fig.set_figwidth(20)
# First plot to the Original time series
axes[0].plot(col.XRP_price, label='Original', color =
    GLOBAL_BLUE, linewidth = 3)
axes[0].legend(loc='upper_left');
# second plot to be for trend
axes[1].plot(trend_estimate, label='Trend', color =
    GLOBAL_RED, linewidth = 3)
axes[1].legend(loc='upper_left');
# third plot to be Seasonality component
axes[2].plot(seasonal_estimate, label='Seasonality', color =
    GLOBAL_YELLOW, linewidth = 3)
axes[2].legend(loc='upper_left');
# last last plot to be Residual component
axes[3].plot(residual_estimate, label='Residuals', color =
    GLOBAL_GREEN, linewidth = 3)
axes[3].legend(loc='upper_left');
plt.savefig('images/' + timeframe + '_decomposition.pdf',
    format = "pdf", dpi = 300, bbox_inches='tight')
plt.savefig('images/' + timeframe + '_decomposition.jpg',
    format = "jpg", dpi = 300, bbox_inches='tight')
return trend_estimate, seasonal_estimate, residual_estimate

def plot_and_save_residuals(residuals, timeframe):
    plt.figure(figsize=(7, 5))
    ax = sns.distplot(residuals, bins = 30)
    ax.set(xlabel='Residual', ylabel='Count')
    plt.axvline(residuals.mean(), c='k', ls='-', lw=2.5)
    plt.axvline(residuals.median(), c='orange', ls='--', lw=2.5)
    plt.savefig('images/' + timeframe + '_residuals_distribution
        .pdf', format = "pdf", dpi = 300, bbox_inches='tight')
    plt.savefig('images/' + timeframe + '_residuals_distribution
        .jpg', format = "jpg", dpi = 300, bbox_inches='tight')
    plt.show()

def test_normality(data):
    totest = data.copy()
    totest = totest.dropna()
    test, p = stats.shapiro(totest)
    print("P-value: " + str(round(p, 3)), " - Statistic: " + str(round(test,
        3)))

def plot_and_do_forecast(filename, xlabel, title, timeframe):
    xrp = pd.read_csv(filename, parse_dates = ['time'])[['time',
        'XRP_price']]
    series = TimeSeries.from_dataframe(xrp, 'time', 'XRP_price')
    split = int(xrp.shape[0]*0.7)
    train, val = series[:split], series[split:]

    model = AutoARIMA()
    model.fit(train)
    prediction = model.predict(len(val))

```

```

fig , ax = plt.subplots(1, 1, figsize = (15, 5))
series.plot(label='XRP_kaina')
prediction.plot(label='Prognoz' , low_quantile=0.05,
                 high_quantile=0.95)
ax.set_title(title , fontsize=22)
ax.set_ylabel('Kaina' , fontsize=10)
ax.set_xlabel(xlabel , fontsize=10)
ax.legend(prop={'size': 20})
ax.grid()
plt.legend()
plt.savefig('images/' + timeframe + '_arima_results.pdf',
            format = "pdf" , dpi = 300,bbox_inches='tight')
plt.savefig('images/' + timeframe + '_arima_results.jpg',
            format = "jpg" , dpi = 300,bbox_inches='tight')
plt.show()
pred = prediction.values().reshape(-1,)
real = val.values().reshape(-1,)
print("RMSE: " ,round(np.sqrt(mean_squared_error(real , pred))) ,
      4))
print("MAE: " ,round(mae(real , pred) , 4))
print("MAPE: " ,round(mean_absolute_percentage_error(real ,
           pred) , 4))

# # Day

# In[8]:


day = pd.read_csv("data/full_day.csv" , parse_dates=['time']) [[
    'time' , 'XRP_price']] .set_index('time')
test_stationarity(day.XRP_price)

# In[9]:


day_trend , day_seasonal , day_residuals = plot_decomposition(day ,
14, "day")

# In[10]:


plot_and_save_residuals(day_residuals , "day")
test_normality(day_residuals)

# In[11]:


plot_and_do_forecast("data/full_day.csv" , "Diena" , "Dienin "
                     "prognoz" , "day")

```

```

# # Hour

# In[12]:


hour = pd.read_csv("data/full_hour.csv", parse_dates=['time']) [[
    'time', 'XRP_price']].set_index('time')
test_stationarity(hour.XRP_price)

# In[13]:


hour_trend, hour_seasonal, hour_residuals = plot_decomposition(
    hour, 12, "hour")

# In[14]:


plot_and_save_residuals(hour_residuals, "hour")
test_normality(hour_residuals)

# In[15]:


plot_and_do_forecast("data/full_hour.csv", "Valanda", "Valandin
    -prognoz", "hour")

# # Minute

# In[16]:


minute = pd.read_csv("data/full_minute.csv", parse_dates=['time'])
[[ 'time', 'XRP_price']].set_index('time')
test_stationarity(minute.XRP_price)

# In[17]:


minute_trend, minute_seasonal, minute_residuals =
    plot_decomposition(minute, 30, "minute")

# In[18]:


plot_and_save_residuals(minute_residuals, "minute")
test_normality(minute_residuals)

```

```
# In[19]:
```

```
plot_and_do_forecast("data/full_minute.csv", "Minut", "  
Minuntin -prognoz", "minute")  
# Takes too much time, can be found in images/  
minute_arima_results.pdf
```

---

### 10.3 Sentimento modelis

```
#!/usr/bin/env python  
# coding: utf-8
```

```
# In[1]:
```

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
plt.style.use('seaborn-whitegrid')  
import re  
from tqdm import tqdm  
import seaborn as sns  
sns.set_style("whitegrid")  
import warnings  
warnings.filterwarnings('ignore')  
from transformers import AutoTokenizer,  
    AutoModelForSequenceClassification, pipeline  
from transformers import RobertaForSequenceClassification,  
    RobertaTokenizer  
import emoji  
  
# Global colors  
GOOGLE_BLUE = "#4285F4"  
GOOGLE_RED = "#DB4437"  
GOOGLE_YELLOW = "#F4B400"  
GOOGLE_GREEN = "#0F9D58"  
  
import emoji  
  
def get_label(text):  
    label = pipe(text)[0]['label']  
    if label == "LABEL_1":  
        return 1  
    return 0  
  
def get_embeddings(text):  
    encoded_input = tokenizer(text, return_tensors='pt',  
        truncation=True, max_length=280)  
    output = model(**encoded_input)
```

```

states = output.hidden_states[-1]
means = np.nanmean(states.detach().numpy(), axis = 1) .
    reshape(1, 768)
meansdf = pd.DataFrame(means).add_prefix("state_")
meansdf["text"] = text
return meansdf

def is_letters_in_text(text):
    return any(c.isalpha() for c in text)

def remove_unused(text):
    new_text = []

    for t in text.split(" "):
        t = '' if t.startswith('@') and len(t) > 1 else t
        t = '' if t.startswith('http') else t
        t = '' if t.startswith('$') else t

        new_text.append(t)

    result = " ".join(new_text).strip()
    if result == "" or not is_letters_in_text(result):
        return np.nan

    return result

def process_text(texts):

    # remove URLs
    texts = re.sub(r'https?://\S+', "", texts)
    texts = re.sub(r'www.\S+', "", texts)
    # remove '
    texts = texts.replace('\'', '')
    # remove symbol names
    texts = re.sub(r'(\#)(\S+)', r'hashtag_\2', texts)
    texts = re.sub(r'(\$)([A-Za-z]+)', r'cashtag_\2', texts)
    # remove usernames
    texts = re.sub(r'(@)(\S+)', r'mention_\2', texts)
    # demojize
    texts = emoji.demojize(texts, delimiters=(" ", " "))

    return texts.strip()

MODELPATH = f"zhayunduo/roberta-base-stocktwits-finetuned"
tokenizer = RobertaTokenizer.from_pretrained(MODELPATH)
model = RobertaForSequenceClassification.from_pretrained(
    MODELPATH, output_hidden_states = True)
pipe = pipeline("text-classification", model=model, tokenizer=
    tokenizer)

# In [6]:

```

```

testtext = "bitcoin_to_the_moon!!!"

# In[7]:
get_label(testtext)

# In[8]:
get_embeddings(testtext)

##### Sentiment
full_tweets = pd.read_csv("raw_data/tweets.csv")
full_tweets["text"] = [remove_unused(i) for i in full_tweets.text]
full_tweets = full_tweets.dropna(axis=0)
full_tweets["text"] = [process_text(i) for i in full_tweets.text]
full_tweets = full_tweets.dropna(axis=0)
full_tweets = full_tweets.reset_index(drop=True)# Takes 32 minutes
embeds = get_embeddings(full_tweets.text[0])
for t in tqdm(full_tweets.text[1:]):
    embeds = pd.concat([embeds, get_embeddings(t)])
embeds = embeds.reset_index(drop=True)# Takes 26 minutes
labels = []
for i in tqdm(full_tweets.text):
    labels.append(get_label(i))
full_tweets["labels"] = labels
full_tweets = full_tweets.reset_index(drop=True)
full = pd.concat([full_tweets, embeds], axis=1)
full.to_csv("data/full_tweets.csv")
# In[10]:
full_tweets = pd.read_csv("data/full_tweets.csv")

# In[11]:
full_tweets

##### Sentiment histogram
# In[12]:

```

```

plt.figure(figsize=(7, 5))
ax = sns.barplot(data=full_tweets.groupby(['labels']).size().
    reset_index(name='counts'),
                  x="labels",
                  y ="counts",
                  color=GOOGLE_BLUE)
ax.set(xlabel='Sentiment', ylabel='Count')
ax.set_xticklabels(['NEG', 'POS'])
ax.bar_label(ax.containers[0])
plt.savefig('images/sentiment_barchart.pdf', format = "pdf", dpi = 300, bbox_inches='tight')
plt.savefig('images/sentiment_barchart.jpg', format = "jpg", dpi = 300, bbox_inches='tight')
plt.show()

```

# ##### Aggregation

# In [13]:

```

tosum = full_tweets[['time', 'labels']]
tosum["time"] = pd.to_datetime(tosum.time, utc = True)
tosum["min"] = tosum.time.dt.ceil("min")
tosum["min"] = tosum["min"].dt.tz_localize(None)
tosum["hour"] = tosum.time.dt.ceil("H")
tosum["hour"] = tosum["hour"].dt.tz_localize(None)
tosum["day"] = tosum.time.dt.ceil("D")
tosum["day"] = tosum["day"].dt.tz_localize(None)
tosum["pos"] = [1 if i == 1 else 0 for i in tosum.labels]
tosum["neg"] = [1 if i == 0 else 0 for i in tosum.labels]

```

# In [14]:

`tosum.head()`

# In [15]:

```

def calc_dom(row):
    if row.pos >= row.neg:
        return "Positive"
    return "Negative"

def add_domination(dataset):
    df = dataset.copy()
    df["dom"] = [calc_dom(i) for i in df.itertuples()]
    return df

```

```
# ## Minute
```

```
# In [16]:
```

```
minutesum = tosum.groupby('min').sum()
minutesum = minutesum.reset_index()
minutesum = add_domination(minutesum)
minutesum['total'] = minutesum.pos + minutesum.neg
minuteprices = pd.read_csv("raw_data/minute.csv")
minuteprices.time = pd.to_datetime(minuteprices.time,
    infer_datetime_format=True, utc=True)
minuteprices.time = minuteprices.time.dt.tz_localize(None)

print("Shape_tweets_join:", minutesum.shape)
print("Shape_prices_join:", minuteprices.shape)
minutedf = pd.merge(minuteprices, minutesum, left_on="time",
    right_on="min", how='left')
print("Shape_after_join:", minutedf.shape)
minutedf = minutedf.drop(['min'], axis=1)
minutedf = minutedf.fillna(0)
minutedf.to_csv("data/full_minute.csv", index=False)
```

```
# In [17]:
```

```
plt.figure(figsize=(15, 7))
ax = sns.scatterplot(data=minutesum,
                      x="min",
                      y="total",
                      hue="dom",
                      palette=[GOOGLE_GREEN, GOOGLE_RED],
                      s=70)
ax.legend(title="Domination", frameon=True)

ax.set(xlabel='Minute', ylabel='Count')
plt.savefig('images/minute_scatter_by_domination.pdf', format="pdf",
            dpi=300, bbox_inches='tight')
plt.savefig('images/minute_scatter_by_domination.jpg', format="jpg",
            dpi=300, bbox_inches='tight')
plt.show()
```

```
# #### Hour
```

```
# In [18]:
```

```
hoursum = tosum.groupby('hour').sum()
hoursum = hoursum.reset_index()
hoursum = add_domination(hoursum)
```

```

hoursum[ "total" ] = hoursum.pos + hoursum.neg

hourprices = pd.read_csv( "raw_data/hour.csv" )
hourprices.time = pd.to_datetime(hourprices.time,
    infer_datetime_format = True, utc = True)
hourprices.time = hourprices.time.dt.tz_localize(None)
print("Tweets_shape:", hoursum.shape)
print("Prices_shape:", hourprices.shape)
hourdf = pd.merge(hourprices, hoursum, left_on="time", right_on=
    "hour", how = 'left')
print("Shape_after_join:", hourdf.shape)
hourdf = hourdf.drop([ 'hour' ], axis = 1)
hourdf = hourdf.fillna(0)
hourdf.to_csv( "data/full_hour.csv" , index = False)

```

# In [19]:

```

plt.figure(figsize=(15, 7))
ax = sns.scatterplot(data=hoursum ,
                      x="hour",
                      y="total",
                      hue = "dom",
                      palette = [GOOGLE_GREEN, GOOGLE_RED],
                      s = 70
                     )
ax.legend(title = "Domination", frameon = True)

ax.set(xlabel='Hour', ylabel='Count')
plt.savefig('images/hour_scatter_by_domination.pdf', format =
    "pdf", dpi = 300,bbox_inches='tight')
plt.savefig('images/hour_scatter_by_domination.jpg', format =
    "jpg", dpi = 300,bbox_inches='tight')
plt.show()

```

# ### Day

# In [20]:

```

daysum = tosum.groupby( 'day' ).sum()
daysum = daysum.reset_index()
daysum = add_domination(daysum)
daysum[ "total" ] = daysum.pos + daysum.neg

dayprices = pd.read_csv( "raw_data/day.csv" )
dayprices.time = pd.to_datetime(dayprices.time,
    infer_datetime_format = True, utc = True)
dayprices.time = dayprices.time.dt.tz_localize(None)
print("Tweets_shape:", daysum.shape)
print("Prices_shape:", dayprices.shape)

```

```

daydf = pd.merge(dayprices , daysum , left_on="time" , right_on="day" , how = 'left')
print("Shape_after_join:" , daydf .shape)
daydf = daydf .drop([ 'day'] , axis = 1)
daydf = daydf .fillna(0)
daydf.to_csv("data/full_day.csv" , index = False)

```

# In[21]:

```

plt . figure(figsize=(15, 7))
ax = sns . scatterplot(data=daysum ,
                       x="day" ,
                       y="total" ,
                       hue = "dom" ,
                       palette = [GOOGLE_GREEN] ,
                       s = 70
                      )
ax.legend(title = "Domination" , frameon = True)

ax.set(xlabel='Day' , ylabel='Count')
plt . savefig('images/day_scatter_by_domination.pdf' , format = "pdf" , dpi = 300 , bbox_inches='tight')
plt . savefig('images/day_scatter_by_domination.jpg' , format = "jpg" , dpi = 300 , bbox_inches='tight')

plt . show()

```

---

## 10.4 Duomenų paruošimas

```

#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
from sklearn.model_selection import train_test_split


def addlag(data , col , maxlag):
    df = data .copy()
    for i in range(1 , maxlag +1):
        df[col + "_L" + str(i)] = df[col].shift(i)

    return df


def read_and_clean_data(filename):
    df = pd.read_csv(filename , parse_dates = [ 'time'])
    maxlag = 4
    df["pospercent"] = df.pos / df.total
    df.total = df.total.astype(int)

```

```

df[ "BTC_change" ] = df.BTC_price.pct_change()
df = addlag(df, "BTC_change", maxlag)
df[ "SPY_change" ] = df.SPY_price.pct_change()
df = addlag(df, "SPY_change", maxlag)
df[ "VXX_change" ] = df.VXX_price.pct_change()
df = addlag(df, "VXX_change", maxlag)
df[ "XAU_change" ] = df.XAU_price.pct_change()
df = addlag(df, "XAU_change", maxlag)
df[ "change" ] = df.XRP_price.pct_change().shift(-1)
df = addlag(df, "change", maxlag)

df = df.drop(['time', 'XRP_price', 'BTC_price', 'SPY_price',
              'VXX_price', 'XAU_price', 'VXX_change', 'labels', 'pos', 'neg',
              'dom'], axis=1)
df = df.dropna()
df = df.reset_index(drop=True)

new_cols = [col for col in df.columns if col != 'change'] + 
           ['change']
df = df[new_cols]

return df

def split_and_save(filename):
    df = read_and_clean_data(filename)
    train, test = train_test_split(df, train_size=0.7, shuffle=False)
    train, valid = train_test_split(train, train_size=0.8,
                                    shuffle=False)
    train = train.reset_index(drop=True)
    valid = valid.reset_index(drop=True)
    test = test.reset_index(drop=True)
    print("Train: ", train.shape, "Valid: ", valid.shape, "Test",
           :, test.shape)
    if "day" in filename:
        add = "day"
    if "hour" in filename:
        add = "hour"
    if "minute" in filename:
        add = "minute"

    train.to_csv("data/" + add + "_train.csv", index=False)
    valid.to_csv("data/" + add + "_valid.csv", index=False)
    test.to_csv("data/" + add + "_test.csv", index=False)

# ## Day

# In [2]:

split_and_save("data/full_day.csv")

```

```

# ## Hour

# In [3]:


split_and_save( "data/full_hour.csv" )

# ## Minute

# In [4]:


split_and_save( "data/full_minute.csv" )

```

---

## 10.5 Tiesinė regresija

```

#!/usr/bin/env python
# coding: utf-8


# In [1]:


import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_squared_log_error as msle
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

def read_data(timeframe):
    train = pd.read_csv("data/" + timeframe + "_train.csv")
    valid = pd.read_csv("data/" + timeframe + "_valid.csv")
    test = pd.read_csv("data/" + timeframe + "_test.csv")

    return train, valid, test

def xy(df):
    return df.iloc[:, :-1], df.iloc[:, -1]

def get_metrics(preds):
    print("MAE: ", mae(preds["True"], preds.Prediction))
    print("RMSE: ", mse(preds["True"], preds.Prediction, squared = False))

# ## Day

# In [2]:

```

```
daytrain , dayvalid , daytest = read_data( "day" )
daytrainx , daytrainy = xy(daytrain)
dayvalidx , dayvalidy = xy(dayvalid)
daytestx , daytesty = xy(daytest)
```

# In [3]:

```
daymodel = sm.OLS(daytrainy , daytrainx).fit()
print(daymodel.summary())
```

# In [4]:

```
pred = pd.DataFrame({ "Prediction" : daymodel.predict(daytestx) , "
    True" : daytesty })
plt.figure(figsize=(15, 7))
ax = pred.plot(figsize=(15, 7))
ax.set(xlabel='Index' , ylabel='Change')
plt.savefig('images/day_linreg_plot.pdf' , format = "pdf" , dpi =
    300,bbox_inches='tight')
plt.savefig('images/day_linreg_plot.jpg' , format = "jpg" , dpi =
    300,bbox_inches='tight')
plt.show()
```

# In [5]:

```
get_metrics(pred)
```

# ## Hour

# In [2]:

```
hourtrain , hourvalid , hourtest = read_data( "hour" )
hourtrainx , hourtrainy = xy(hourtrain)
hourvalidx , hourvalidy = xy(hourvalid)
hourtestx , hourtesty = xy(hourtest)
```

# In [3]:

```
hourmodel = sm.OLS(hourtrainy , hourtrainx).fit()
print(hourmodel.summary())
```

```

# In[4]:


pred = pd.DataFrame({ "Prediction" : hourmodel.predict(hourtestx) ,
                      "True" : hourtesty })
plt.figure(figsize=(15, 7))
ax = pred.plot(figsize=(15, 7))
ax.set(xlabel='Index', ylabel='Change')
plt.savefig('images/hour_linreg_plot.pdf', format = "pdf", dpi =
            300,bbox_inches='tight')
plt.savefig('images/hour_linreg_plot.jpg', format = "jpg", dpi =
            300,bbox_inches='tight')
plt.show()

# ## Minute

# In[22]:


minutetrain, minutevalid, minutetest = read_data("minute")
minutetrainx, minutetrainy = xy(minutetrain)
minutevalidx, minutevalidy = xy(minutevalid)
minutetestx, minutetesty = xy(minutetest)

# In[23]:


minutemodel = sm.OLS(minutetrainy, minutetrainx).fit()
print(minutemodel.summary())


# In[24]:


minutetesty.shape

# In[33]:


plt.plot

# In[37]:


pred = pd.DataFrame({ "Prediction" : minutemodel.predict(
                      minutetestx), "True" : minutetesty })
plt.figure(figsize=(15, 7))

```

```

ax = pred[['True', 'Prediction']].plot(figsize=(15, 7), color =
    ['tab:orange', 'tab:blue'])
ax.set(xlabel='Index', ylabel='Change')
plt.savefig('images/minute_linreg_plot.pdf', format = "pdf", dpi
    = 300,bbox_inches='tight')
plt.savefig('images/minute_linreg_plot.jpg', format = "jpg", dpi
    = 300,bbox_inches='tight')
plt.show()

```

# In[13]:

---

```
get_metrics(pred)
```

---

## 10.6 Pycaret paketas

---

```
#!/usr/bin/env python
# coding: utf-8
```

# In[1]:

```

from pycaret.regression import *
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(555)

def read_data(timeframe):
    train = pd.read_csv("data/" + timeframe + "_train.csv")
    valid = pd.read_csv("data/" + timeframe + "_valid.csv")
    test = pd.read_csv("data/" + timeframe + "_test.csv")

    return train, valid, test

```

# ## Day

# In[5]:

```
day_train, day_valid, day_test = read_data("day")
```

# In[6]:

```
sday = setup(data = day_train,
              target = 'change',
              silent = True,
```

```

        numeric_features = [ "total" ] ,
session_id = 1 ,
test_data = day_valid)
set_config( 'seed' , 555)

# In[7]:


mday = compare_models()

# In[8]:


plot_model(mday)

# In[9]:


plot_model(mday, plot = 'error')

# In[10]:


plot_model(mday, plot='feature')

# In[11]:


pred = predict_model(mday, data = day_test)[[ 'change' , 'Label' ]]
pred = pred.rename({ 'change' : 'True' , 'Label' : 'Prediction' } , axis
= 1)
plt.figure(figsize=(15, 7))
ax = pred.plot(figsize=(15, 7))
ax.set(xlabel='Index' , ylabel='Change')
plt.savefig('images/day_pycaret_plot.pdf' , format = "pdf" , dpi =
300,bbox_inches='tight')
plt.savefig('images/day_pycaret_plot.jpg' , format = "jpg" , dpi =
300,bbox_inches='tight')
plt.show()

# In[12]:


plot_model(mday, save = True)
plot_model(mday, plot = 'error' , save = True)
plot_model(mday, plot='feature' , save = True)

```

```

# ## Hour

# In[13]:


hour_train, hour_valid, hour_test = read_data("hour")

# In[14]:


shour = setup(data = hour_train,
              target = 'change',
              silent = True,
              numeric_features = ["total"],
              session_id = 2,
              test_data = hour_valid)
set_config('seed', 555)

# In[15]:


mhour = compare_models()

# In[16]:


plot_model(mhour)

# In[17]:


plot_model(mhour, plot = "error")

# In[18]:


plot_model(mhour, plot = "feature")

# In[19]:


pred = predict_model(mhour, data = hour_test)[['change', 'Label']]
pred = pred.rename({'change': 'True', 'Label': 'Prediction'}, axis = 1)
plt.figure(figsize=(15, 7))
ax = pred.plot(figsize = (15, 7))
ax.set(xlabel='Index', ylabel='Change')

```

```
plt.savefig('images/hour_pycaret_plot.pdf', format = "pdf", dpi = 300,bbox_inches='tight')
plt.savefig('images/hour_pycaret_plot.jpg', format = "jpg", dpi = 300,bbox_inches='tight')
plt.show()
```

# In[20]:

```
plot_model(mhour, save = True)
plot_model(mhour, plot = "error", save = True)
plot_model(mhour, plot = "feature", save = True)
```

# ## Minute

# In[21]:

```
minute_train, minute_valid, minute_test = read_data("minute")
```

# In[22]:

```
shour = setup(data = minute_train,
              target = 'change',
              silent = True,
              numeric_features = ["total"],
              session_id = 3,
              test_data = minute_valid)
set_config('seed', 555)
```

# In[23]:

```
mminute = compare_models()
```

# In[24]:

```
plot_model(mminute)
```

# In[25]:

```
plot_model(mminute, plot = "error")
```

# In[26]:

```

plot_model(mminute, plot = "feature")

# In[27]:


pred = predict_model(mminute, data = minute_test)[['change', 'Label']]
pred = pred.rename({'change': 'True', 'Label': 'Prediction'}, axis = 1)
plt.figure(figsize=(15, 7))
ax = pred.plot(figsize=(15, 7))
ax.set(xlabel='Index', ylabel='Change')
plt.savefig('images/minute_pycaret_plot.pdf', format = "pdf",
            dpi = 300,bbox_inches='tight')
plt.savefig('images/minute_pycaret_plot.jpg', format = "jpg",
            dpi = 300,bbox_inches='tight')
plt.show()

# In[28]:


plot_model(mminute, save = True)
plot_model(mminute, plot = "error", save = True)
plot_model(mminute, plot = "feature", save = True)

```

---

## 10.7 LSTM modelis

---

```

#!/usr/bin/env python
# coding: utf-8

# In[24]:


import os
import datetime

import IPython
import IPython.display
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(555)
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Dropout,
        LSTM

```

```

import matplotlib.pyplot as plt
mpl.rcParams['figure.figsize'] = (15, 7)
mpl.rcParams['axes.grid'] = True
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae

def read_data(timeframe):
    train = pd.read_csv("data/" + timeframe + "_train.csv")
    valid = pd.read_csv("data/" + timeframe + "_valid.csv")
    test = pd.read_csv("data/" + timeframe + "_test.csv")

    return train, valid, test

def get_windowed_data(tempdf, wsize):
    f = []
    l = []
    for i in range(wsize, tempdf.shape[0]):
        window = tempdf.iloc[i-wsize:i, :]
        features = window.iloc[:, :-1].values
        labels = window.iloc[-1, -1]
        f.append(features)
        l.append(labels)

    f = np.array(f)
    l = np.array(l).reshape(len(l), 1)
    return f, l

def get_train_valid_test_x_y_data(train, valid, test, timeframe):
    if timeframe == "day":
        wsize = 2
    if timeframe == "hour":
        wsize = 12
    if timeframe == "minute":
        wsize = 30
    trainx, trainy = get_windowed_data(train, wsize)
    print("TRAIN: ", "x:", trainx.shape, "y:", trainy.shape)
    validx, validy = get_windowed_data(valid, wsize)
    print("VALID: ", "x:", validx.shape, "y:", validy.shape)
    testx, testy = get_windowed_data(test, wsize)
    print("TEST: ", "x:", testx.shape, "y:", testy.shape)

    return trainx, trainy, validx, validy, testx, testy

def build_lstm_model(input_data, output_size, neurons,
                     activ_func='linear',
                     dropout=0.2, loss='mse', optimizer='adam'):
    model = Sequential()
    model.add(LSTM(neurons, input_shape=(input_data.shape[1],
                                         input_data.shape[2])))
    model.add(Dropout(dropout))
    model.add(Dense(units=1))
    model.add(Activation(activ_func))

```

```

model.compile(loss=loss, optimizer=optimizer)
return model

def analyze_predictions(model, testx, testy, timeframe):
    preds = pd.DataFrame({ "Prediction":model.predict(testx).
        squeeze(), "True": testy.squeeze() })
    print("MAE: ", mae(preds["True"], preds.Prediction))
    print("RMSE: ", mse(preds["True"], preds.Prediction, squared
        = False))
    ax = preds.plot(figsize = (15, 7))
    ax.set(xlabel='Index', ylabel='Change')
    plt.savefig('images/' +timeframe+'_lstm_results.pdf', format
        = "pdf", dpi = 300,bbox_inches='tight')
    plt.savefig('images/' +timeframe+'_lstm_results.jpg', format
        = "jpg", dpi = 300,bbox_inches='tight')
    plt.show()

def xy(df):
    return df.iloc[:, :-1], df.iloc[:, -1]

def plot_errors(modelfit, timeframe):
    errors = pd.DataFrame({ 'Training_loss':modelfit.history['
        loss'], 'Validation_loss': modelfit.history['val_loss']})
    ax = errors.plot()
    plt.title('LSTM')
    plt.xlabel('Epochs_numbers')
    plt.ylabel('MSE_numbers')
    plt.legend()
    plt.savefig('images/' +timeframe+'_lstm_errors.pdf', format =
        "pdf", dpi = 300,bbox_inches='tight')
    plt.savefig('images/' +timeframe+'_lstm_errors.jpg', format =
        "jpg", dpi = 300,bbox_inches='tight')
    plt.show()

zero_base = True
lstm_neurons = 50
epochs = 200
batch_size = 64
loss = 'mse'
dropout = 0.24
optimizer = 'adam'

# # Day

# In[18]:


daytrain, dayvalid, daytest = read_data("day")
daytrainx, daytrainy, dayvalidx, dayvalidy, daytestx, daytesty =
    get_train_valid_test_x_y_data(daytrain, dayvalid, daytest, "day")

```

```
# In[19]:
```

```
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                       patience=5)

model = build_lstm_model(
    daytrainx, output_size=1, neurons=lstm_neurons, dropout=
        dropout, loss=loss,
    optimizer=optimizer)

modelfit = model.fit(
    daytrainx, daytrainy, validation_data=(dayvalidx, dayvalidy)
    , epochs=epochs, batch_size=batch_size, verbose=2,
    shuffle=True, callbacks = [es])
```

```
# In[25]:
```

```
plot_errors(modelfit, "day")
```

```
# In[27]:
```

```
analyze_predictions(model, daytestx, daytesty, "day")
```

```
# # Hour
```

```
# In[28]:
```

```
hourtrain, hourvalid, hourtest = read_data("hour")
hourtrainx, hourtrainy, hourvalidx, hourvalidy, hourtestx,
hourtesty = get_train_valid_test_x_y_data(hourtrain,
    hourvalid, hourtest, "hour")
```

```
# In[29]:
```

```
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                       patience=5)

model = build_lstm_model(
    hourtrainx, output_size=1, neurons=lstm_neurons, dropout=
        dropout, loss=loss,
    optimizer=optimizer)

modelfit = model.fit(
    hourtrainx, hourtrainy, validation_data=(hourvalidx,
    hourvalidy), epochs=epochs, batch_size=batch_size,
```

```

verbose=2, shuffle=True, callbacks = [es])

# In[30]:

plot_errors(modelfit, "hour")

# In[31]:

analyze_predictions(model, hourtestx, hourtesty, "hour")

## Minute

# In[32]:

mtrain, mvalid, mtest = read_data("minute")
mtrainx, mtrainy, mvalidx, mvalidy, mtestx, mtesty =
    get_train_valid_test_x_y_data(mtrain, mvalid, mtest, "minute")

# In[33]:

es = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                       patience=5)

model = build_lstm_model(
    mtrainx, output_size=1, neurons=lstm_neurons, dropout=
        dropout, loss=loss,
    optimizer=optimizer)

modelfit = model.fit(
    mtrainx, mtrainy, validation_data=(mvalidx, mvalidy), epochs
        =epochs, batch_size=batch_size, verbose=2, shuffle=True,
    callbacks = [es])

# In[34]:

plot_errors(modelfit, "minute")

# In[35]:


analyze_predictions(model, mtestx, mtesty, "minute")

```

---

## Literatūra

- [ABR17] Nashirah Abu Bakar and Sofian Rosbi. Autoregressive integrated moving average (arima) model for forecasting cryptocurrency exchange rate in high volatility environment: A new insight of bitcoin transaction. *International Journal of Advanced Engineering Research and Scinece*, 4, 11 2017.
- [AEK<sup>+</sup>21] M. Eren Akbiyik, Mert Erkul, Killian Kaempf, Vaiva Vasiliauskaitė, and Nino Antulov-Fantulin. Ask "who", not "what": Bitcoin volatility forecasting with twitter data, 2021.
- [AHNI18] Jethin Abraham, Danny W. Higdon, Johnny Nelson, and Juan Ibarra. Cryptocurrency price prediction using tweet volumes and sentiment analysis. 2018.
- [Ali20] Moez Ali. *PyCaret: An open source, low-code machine learning library in Python*, April 2020. PyCaret version 1.0.
- [SJOL20] Otabek Sattarov, Heung Jeon, Ryumduck Oh, and Jun Lee. Forecasting bitcoin price fluctuation by twitter sentiment analysis. pages 1–4, 11 2020.
- [SN18] Sima Siami-Namini and Akbar Siami Namin. Forecasting economics and financial time series: ARIMA vs. LSTM. *CoRR*, abs/1803.06386, 2018.